

IMPROVING IMAGING PIPELINES BY SENSOR
OPTIMIZATION AND LOW-LEVEL IMAGE
ENHANCEMENT

DISSERTATION
ZUR ERLANGUNG DES GRADES EINES
DOKTORS DER NATURWISSENSCHAFTEN

vorgelegt von
Hendrik Sommerhoff, M.Sc.

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen

Siegen, 2026

Betreuer und erster Gutacher
Prof. Dr. Andreas Kolb
Universität Siegen

Zweiter Gutacher
Prof. Dr. Felix Heide
Princeton University

Tag der mündlichen Prüfung
10. Juli 2026

ABSTRACT

Modern digital imaging pipelines transform incident light from the physical world into high-level semantic information for a variety of applications, such as medical imaging, robotics, visualization, and autonomous driving. They consist of multiple stages, including the focusing of light on an image sensor by optical systems, the measurement on the sensor itself, and further processing by an image signal processor (ISP) that also reduces noise and other errors from the measurement process, before the image is consumed by a task-specific algorithm such as a deep neural network. Conventionally, the training of the network is largely decoupled from the rest of the components, which are typically optimized for general-purpose use cases or human perception. This thesis investigates how imaging pipelines can be improved (i) as a whole by jointly optimizing sensor parameters and downstream learning-based algorithms, and (ii) by developing efficient low-level reconstruction and error-removal methods to handle the unavoidable imperfections introduced in the measurement process.

The first part of this thesis focuses on the task-driven design of sensors. It develops methods to optimize the physical position of pixels on a sensor and the shape of spectral color filters, jointly with a downstream neural network to solve specific tasks. Although learned sensor layouts show a significant improvement in a variety of tasks compared to uniform grids, no such improvement could be found for learned spectral color filters in experiments for the specific task of semantic segmentation in autonomous driving, suggesting that, in this setting, the information gained from the spatial positioning of pixels is more impactful than fine-grained spectral information.

The second part of the thesis improves the handling of imperfect measurements through several methods. It includes a provably convergent plug-and-play variational scheme with learned priors for solving inverse problems, a generic depth image enhancement method utilizing normal consistency constraints, and an outlier-robust point-based rendering algorithm that can directly render very noisy point clouds produced by depth sensors.

ZUSAMMENFASSUNG

Moderne digitale Bildgebungsverfahren transformieren einfallendes Licht aus der realen Welt in semantische Informationen für eine Vielzahl von Anwendungen, wie in der medizinischen Bildgebung, der Robotik, der Visualisierung und dem autonomen Fahren. Diese Bilderverarbeitungs-Pipelines bestehen aus mehreren Stufen, darunter die Fokussierung des Lichts auf einen Bildsensor durch optische Systeme, die Messung auf dem Sensor selbst, sowie die Verarbeitung durch einen Image Signal Processor (ISP), der unter anderem Rauschen und andere Fehler aus dem Messprozess reduziert, bevor das resultierende Bild von einem Algorithmus (wie einem neuronalen Netz) weiterverarbeitet wird. Normalerweise ist das Training des Netzes weitgehend von den anderen Komponenten entkoppelt. Diese werden oft für allgemeine Anwendungsfälle oder für die menschliche Wahrnehmung optimiert. Diese Arbeit untersucht, wie solche Pipelines in der Bildverarbeitung (i) als Ganzes durch die gemeinsame Optimierung von Sensorparametern und lernbasierten Algorithmen verbessert werden können und (ii) wie effiziente Rekonstruktions- und Korrekturverfahren entwickelt werden können, um die unvermeidbaren Fehler in den Bilddaten zu behandeln.

Der erste Teil dieser Arbeit konzentriert sich auf dem Design von Sensoren für spezifische Aufgaben. Es werden Methoden entwickelt, um das Layout von Pixeln auf einem Sensor sowie die Form spektraler Farbfilter gemeinsam mit einem nachgelagerten neuronalen Netz für eine spezifische Aufgabe zu optimieren. Während gelernte Sensorlayouts in einer Vielzahl von Aufgaben gegenüber gleichmäßig verteilten Pixeln deutliche Verbesserungen zeigen, ließ sich ein solcher Vorteil für gelernte Farbfilter durch Experimenten zur semantischen Segmentierung im autonomen Fahren nicht nachweisen. Dies deutet darauf hin, dass in diesem Szenario die durch die räumliche Anordnung der Pixel gewonnene Information mehr Einfluss hat als feiner aufgelöste spektrale Information.

Der zweite Teil der Arbeit verbessert die Verarbeitung fehlerbehafteter Bilddaten durch verschiedene Verfahren. Er umfasst ein Plug-and-Play Verfahren mit nachweislicher Konvergenz und gelernten Regularisierern zur Lösung inverser Probleme, eine generische Methode zur Verbesserung von Tiefenbildern sowie einen punktbasierten Renderalgorithmus, zum Rendern von Punktwolken mit starkem Rauschen.

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my advisor, Prof. Dr. Andreas Kolb, for his guidance, encouragement and support throughout the years of my PhD, and I am particularly grateful for the freedom he gave me in exploring different research directions and for always supporting my curiosity and ideas.

I would also like to thank all present and former members of the Computer Graphics group, the Computer Vision group, and the Learning to Sense research unit for the many valuable discussions and their support during difficult times.

Last but not least, I want to thank my family and friends for their patience and unwavering support throughout my PhD. Thank you for always being there for me, even when I could not spend as much time with you as I would have liked.

PUBLICATIONS

The contributions presented in this dissertation have been published in the following publications. Throughout the thesis, references to these publications will be printed in bold.

- [**Som+24**] H. Sommerhoff, S. Agnihotri, M. Saleh, M. Moeller, M. Keuper, B. Choubey, and A. Kolb. "Task-driven sensor layouts - Joint optimization of pixel layout and network parameters." In: *Proceedings of the IEEE International Conference on Computational Photography (ICCP)*. 2024, pp. 1–10.
- [**SK22**] H. Sommerhoff and A. Kolb. "A generic framework for depth reconstruction enhancement." In: *Journal of Imaging* 8.5 (2022), p. 138.
- [**SK24**] H. Sommerhoff and A. Kolb. "Hashed, binned A-buffer for real-time outlier removal and rendering of noisy point clouds." In: *The Visual Computer* 40.3 (2024), pp. 1825–1838.
- [**SKM19**] H. Sommerhoff, A. Kolb, and M. Moeller. "Energy dissipation with plug-and-play priors." In: *NeurIPS Workshop on Solving Inverse Problems with Deep Networks*. 2019.

CONTENTS

1	INTRODUCTION	1
1.1	Contributions	2
1.2	Thesis Outline	3
2	FOUNDATIONS	5
2.1	Image Formation	5
2.1.1	Optics	6
2.1.2	Measurement	14
2.2	Deep Learning	23
2.2.1	Overview	23
2.2.2	Layers and other building blocks	25
2.2.3	Loss functions	32
2.2.4	Model training and evaluation	33
2.3	Inverse Problems in Imaging	38
2.3.1	Variational methods	39
2.3.2	Deep learning and hybrid methods	42
2.4	Point-based Rendering	43
2.4.1	Point clouds	43
2.4.2	Rendering opaque splats	44
2.4.3	EWA splatting	45
3	LEARNING SENSOR LAYOUTS FOR DEEP LEARNING TASKS	47
3.1	Introduction	47
3.2	Related Work	49
3.2.1	Pixel layout	49
3.2.2	End-to-end optimization of the ISP pipeline	49
3.2.3	Superpixel-based methods	50
3.3	Differentiable Sensor Layout Optimization	51
3.3.1	Measurement equation	52
3.3.2	Parameterizing pixel layouts	53
3.3.3	End-to-end optimization	55
3.4	Implementation Details	57
3.4.1	Numerical integration	57
3.4.2	Integration into existing deep learning pipelines	58
3.4.3	Handling image-to-image tasks	59
3.5	Experiments	59
3.5.1	MNIST classification	59
3.5.2	Semantic segmentation	61
3.5.3	Multi-label classification	66
3.6	Conclusions and Limitations	67
4	LEARNING COLOR FILTERS FOR DEEP LEARNING TASKS	69

4.1	Introduction	69
4.2	Related Work	70
4.3	Method	71
4.3.1	Image formation model	72
4.3.2	Differentiable shot noise	73
4.3.3	Loss functions	74
4.4	Experiments	74
4.4.1	Experiment setup	74
4.4.2	Results	76
4.4.3	Discussion	78
4.5	Conclusions and Future Work	79
5	ENERGY DISSIPATION WITH PLUG-AND-PLAY PRIORS	81
5.1	Introduction	81
5.2	Provably dissipating model-based energies with plug-and-play priors	82
5.3	Results	85
5.4	Conclusions	86
6	DEPTH ENHANCEMENT WITH GEOMETRIC CONSTRAINTS	89
6.1	Introduction	89
6.2	Related work	91
6.3	Method	92
6.3.1	GeoNet	92
6.3.2	General depth enhancement network	94
6.3.3	Loss functions	95
6.3.4	Implementation details	96
6.4	Evaluation	97
6.4.1	Training and tasks	97
6.4.2	Results	98
6.5	Discussion and conclusions	103
7	REAL-TIME RENDERING OF NOISY POINT CLOUDS	105
7.1	Introduction	105
7.2	Related work	107
7.2.1	Point-based rendering	107
7.2.2	Multifragment rendering	108
7.3	Problems of EWA with large outliers	109
7.4	Proposed method	111
7.4.1	Binned A-buffer	111
7.4.2	GPU implementation	112
7.4.3	Spatial Hashing	114
7.4.4	Handling surfaces at bin borders	115
7.5	Experiments and Evaluation	117
7.5.1	Depth reconstruction accuracy	119
7.5.2	Color reconstruction quality	121
7.5.3	High noise and outlier levels	123
7.5.4	Robustness to varying density	124
7.5.5	Performance	125
7.5.6	Ablations with overlapping bins	126

7.6	Conclusions and Limitations	127
8	CONCLUSIONS	129
	BIBLIOGRAPHY	131

LIST OF FIGURES

Figure 2.1	Pinhole Camera Model	7
Figure 2.2	Thin lens model	8
Figure 2.3	Imaging a point out of focus	11
Figure 2.4	Optical aberrations	12
Figure 2.5	Sensor responsivity curve	17
Figure 2.6	Bayer pattern	18
Figure 2.7	Artificial neuron	24
Figure 2.8	Fully connected layer	25
Figure 2.9	Activation Functions	26
Figure 2.10	Transformer encoder	31
Figure 2.11	Deblurring by inversion	38
Figure 3.1	IC layouts for non-uniform pixels	51
Figure 3.2	Sensor layout learning pipeline	52
Figure 3.3	Sensor layout parameterizations	55
Figure 3.4	Quantities for gradient computation	56
Figure 3.5	Learned layout for MNIST classification	60
Figure 3.6	Example segmentations on Cityscapes	62
Figure 3.7	Learned layouts for semantic segmentation	63
Figure 3.8	Per-pixel accuracy gain	66
Figure 4.1	Color filter learning pipeline	71
Figure 4.2	Baseline color filters	75
Figure 4.3	Optimized color filters	76
Figure 5.1	Example of diverging algorithm	83
Figure 5.2	PSNR curves	85
Figure 6.1	Network architecture	95
Figure 6.2	Denoising results	100
Figure 6.3	Deblurring results	101
Figure 6.4	Depth-only super-resolution results	101
Figure 6.6	Normal map visualization	102
Figure 6.5	RGB-guided super-resolution results	102
Figure 7.1	Comparison of EWA and binned A-buffer	110
Figure 7.2	Overlapping bins advantages and disadvantages	116
Figure 7.3	General experiment setup	118
Figure 7.4	Results for the <i>fireplace room</i> scene	119
Figure 7.5	Results for the <i>table scene</i>	120
Figure 7.6	Results for the <i>memorial</i> dataset	122
Figure 7.7	Results on the augmented ICL-NUIM dataset	122

Figure 7.8	Stanford Dragon with extreme outliers	123
Figure 7.9	Evaluation on fireplace room with varying point densities	124
Figure 7.10	Error/recall trade-off	126
Figure 7.11	Examples of Disjoint overlapping bins	127

LIST OF TABLES

Table 3.1	MNIST test set accuracy	61
Table 3.2	Semantic segmentation results on Cityscapes	62
Table 3.3	Ablation of semantic segmentation experiments for different sensor resolutions	65
Table 3.4	Quantitative CelebA results	66
Table 4.1	Segmentation results with 4 channel filters	77
Table 4.2	Repeated experiments	77
Table 5.1	Deblurring results	86
Table 5.2	Super-resolution results	86
Table 6.1	Learnable parameters and timings of used networks	96
Table 6.2	Results on NYU dataset	98
Table 6.3	Results on Middlebury dataset	99
Table 7.1	Quantitative results for <i>fireplace room</i>	120
Table 7.2	Quantitative color reconstruction results (PSNR) on the ICL-NUIM scenes.	122
Table 7.3	Framerate in different scenes	125
Table 7.4	Memory and computational overhead over EWA	125

LISTINGS

Listing 7.1	A-buffer pseudocode	113
Listing 7.2	Hashing scheme with collision handling	114

ACRONYMS

AMCW	Amplitude-modulated continuous wave
CCD	Charge-Coupled Device
CFA	Color Filter Array
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CoC	Circle of Confusion
DOF	Depth of Field
DSNU	Dark Signal Non-Uniformity
EWA	Elliptical Weighted Average
FPGA	Field Programmable Gate Array
GAN	Generative Adversarial Network
HSI	Hyperspectral image
ISP	Image Signal Processor
MAE	Mean Absolute Error
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NIR	Near-infrared
NLM	Non-local means
PMD	Photonic Mixing Device
PRNU	Photo Response Non-Uniformity
PSNR	Peak Signal-to-Noise Ratio
RMSE	Root Mean Square Error
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure
ToF	Time of Flight
TV	Total Variation
ViT	Vision Transformer
VJP	Vector-Jacobian Product

Modern digital imaging systems are the foundation for a wide range of applications, such as photography, medical imaging, robotics, and autonomous driving. These systems transform incident light from the physical world into high-level semantic information suitable for the corresponding downstream task, and consist of multiple components, including optical elements, an image sensor, analog-to-digital converters, and an image signal processor (ISP) (containing algorithms to solve low-level vision tasks such as denoising and demosaicing), followed by a task-specific algorithm or neural network.

In many practical systems, the design of the individual stages of this imaging pipeline is largely decoupled from the task to be solved. Optics and sensors are commonly developed for general-purpose use-cases, with the main criterion being human perception. Because this focus on perception is so common, downstream algorithms are also typically designed with these general-purpose imaging systems in mind. This is especially the case for learning-based techniques, such as neural networks, since they can make use of the large amount of already existing training data. Although these general-purpose systems are modular, i. e., the same camera system can be used with many different tasks, they may be suboptimal when the ultimate goal is the performance on a *specific* computational task and not the perceptual quality of the intermediate images. Thus, a more integrated approach that involves the whole imaging pipeline and considers not only neural network weights as optimization variables but also the configuration parameters of the optics and sensor is a promising research direction.

At the same time, each stage of an imaging pipeline introduces unavoidable errors, such as optical aberrations, noise, and discretization or quantization errors. Many of these error sources have trade-offs that are closely linked to the configuration of the optics and sensor, e. g., larger pixels result in less noise. In many imaging modalities, such as time of flight (ToF) depth cameras under strong ambient lighting conditions, the errors are quite strong and lead to noisy and incomplete depth maps that require processing before they can be reliably used for higher-level tasks. Solution methods for these low-level vision problems, such as denoising, deblurring, and super-resolution, have in recent years been dominated by deep learning techniques. However, physically motivated classical approaches with theoretical guarantees are still attractive in many safety critical applications. Improving the

performance of these low-level vision methods is also a key aspect for improving the overall imaging pipeline.

1.1 CONTRIBUTIONS

This dissertation investigates how imaging pipelines can be improved by (i) jointly optimizing sensor parameters and downstream learning-based algorithms for high-level tasks, and (ii) developing efficient, low-level reconstruction and error-removal methods that account for the underlying noise characteristics and defects, which are unavoidable in image acquisition. Specifically, the thesis makes the following contributions.

TASK-DRIVEN OPTIMIZATION OF SENSOR LAYOUTS A framework for the joint optimization of the spatial arrangement and shape of pixels on an image sensor with a downstream task-specific neural network is introduced. The method includes a differentiable parameterization of non-uniform rectangular and curvilinear pixel geometries, which enables end-to-end optimization by common gradient-based optimization algorithms. Experiments on semantic segmentation for autonomous driving and other tasks show that, for a given pixel budget, tailored sensor layouts can significantly outperform conventional uniform layouts. This approach has been published in [Som+24].

END-TO-END FRAMEWORK FOR LEARNING COLOR FILTERS An end-to-end differentiable sensor simulation is developed for learning spectral color filters for a specific task. The sensor model includes a realistic differentiable shot noise model, and allows the joint optimization of spectral filter curves and downstream networks using hyperspectral images as training data. Under realistic noise assumptions and for the specific task of semantic segmentation in autonomous driving, experiments show that the learned filters do not have a statistically significant advantage over hand-crafted filters.

ENERGY DISSIPATION WITH PLUG-AND-PLAY PRIORS Plug-and-play methods are flexible hybrid algorithmic schemes that combine physically informed, classical variational methods with powerful (learned) denoising priors to solve inverse problems, such as image reconstruction tasks. However, when the denoiser is a deep neural network, either theoretical guarantees of these algorithmic schemes have to be sacrificed, or heavy restrictions have to be put on the network. The thesis contributes a novel algorithmic scheme based on the proximal gradient method, which *provably converges* to the minimizer of a given energy with *any* deep denoising prior, without any restrictions on the network. This method has been published in [SKM19].

GENERIC FRAMEWORK FOR DEPTH ENHANCEMENT A modular deep learning architecture is proposed for the enhancement of depth maps obtained from arbitrary sources. The framework utilizes normal consistency constraints to enforce the inherent geometric relationship between depth and surface normals. The method is applicable to low-level vision tasks such as depth denoising, deblurring and super-resolution, and experiments confirm that inclusion of normal consistency improves reconstruction quality. This method has been published in [SK22].

OUTLIER-ROBUST POINT CLOUD RENDERING A direct point-based rendering technique is presented, which enables real-time visualization of very noisy point clouds. The method employs a novel binned A-buffer data structure that uses a hashing scheme to efficiently accumulate and filter point contributions per pixel, which allows removal of heavy outliers without preprocessing the point clouds. This approach is particularly effective for point clouds produced by noisy depth sensors such as ToF cameras under poor ambient lighting conditions. This method has been published in [SK24].

1.2 THESIS OUTLINE

The remainder of the thesis is structured as follows. [Chapter 2](#) explains mathematical and conceptual foundations needed in this thesis. It begins with the physical and mathematical foundations of optics and the measurement process of complementary metal-oxide-semiconductor (CMOS) and ToF sensors. The explanations in this section focus in particular on the understanding and modeling of imperfections and error sources of the various stages of the image formation process, which are central to this dissertation. The chapter continues with an introduction on deep learning and both classical and hybrid methods to solve inverse problems, which can be utilized for the removal of the imperfections introduced in the image formation process. [Chapter 2](#) finishes with the basic concepts of point cloud representation and rendering. [Chapters 3](#) and [4](#) present data-driven methods for the joint optimization of network parameters and the spatial sensor layout ([Chapter 3](#)) or color filters ([Chapter 4](#)) to solve higher-level vision tasks. The next two chapters deal with low-level vision tasks for improving existing images that have already been measured by a sensor. [Chapter 5](#) explains the provably convergent plug-and-play algorithmic scheme to solve inverse problems with learned priors, and [Chapter 6](#) develops the general depth image enhancement framework using geometric constraints. Then, [Chapter 7](#) introduces a point based rendering method that can deal with high levels of noise and outliers stemming from noisy depth measurements.

Finally, [Chapter 8](#) summarized the key findings of this dissertation and gives possible directions for future work.

2 | FOUNDATIONS

This chapter begins with fundamental concepts and sources of error in the image formation process, such as optics and measurement, in [Section 2.1](#). Afterwards, it explains deep neural networks in [Section 2.2](#), which are used in various parts of this thesis. In [Section 2.3](#) the problem statement and solution methods for inverse problems are introduced. Inverse problems can model common imperfections during image formation and other image reconstruction problems. Finally, [Section 2.4](#) gives a brief overview of point cloud rendering techniques.

2.1 IMAGE FORMATION

Imaging systems convert light from the physical world into digital signals that can be stored, processed, and interpreted. An understanding of this image formation process is crucial to describe the error sources that arise in computer vision and computational imaging tasks [[Sze22](#)]. The physical properties of light (or more generally electromagnetic radiation), the design of the optical elements and sensor characteristics all influence the quality and accuracy of acquired images.

This section introduces the main physical and mathematical concepts behind image formation. [Section 2.1.1](#) revisits relevant principles of geometrical optics, including the behavior of light through lenses, which describes how scenes are projected onto image sensors, as well as optical aberrations. Subsequently, [Section 2.1.2](#) deals with the measurement of light by complementary metal-oxide-semiconductor (CMOS) or charge-coupled device (CCD) sensors. The *measurement equation* is introduced as a mathematical model that describes how the light hitting individual pixels on the image plane is converted into energy measurements. This model serves as a foundation for the later work on the optimization of sensor layouts and color filters in this dissertation. The measurement process is subject to various types of noise sources, which are also modeled in this section. Finally, [Section 2.1.2.6](#) explains depth imaging with time of flight (ToF) sensors and their unique error sources.

2.1.1 Optics

This section will give a brief introduction to the basic optical principles relevant to this dissertation and the role of an optical system in the broader imaging pipeline. A more complete reference can be found in [Hec17] or [Sze22].

2.1.1.1 Overview

Optics is the field of physics that deals with the behavior of light. Light is a special case of electromagnetic radiation, which is a physical phenomenon described as disturbances of the electromagnetic field propagating through space as waves. The wavelength of visible light lies in the range of roughly 380 nm (violet) to 780 nm (red). However, this dissertation follows a more broad use of the term *light* and also includes parts of the near infrared spectrum with a wavelength of up to 950 nm.

Light

A complete physical description of how light behaves and propagates requires an understanding of its wave-like nature, as well as its electromagnetic and quantum properties. This involves solving Maxwell's equations to describe electromagnetic wave propagation and the principles of quantum mechanics to describe phenomena such as photon quantization and fluorescence [Hec17]. However, for most macroscopic imaging applications, such a comprehensive model is both impractical and unnecessary.

Wave optics

A widely used simplification is the *geometric optics approximation*, which treats light as straight rays that only bend at the interface of materials of different refractive indices. This approximation holds when the wavelength of light is much smaller than the size of optical elements and surface structures with which the light interacts.

Geometric optics

An *optical system* refers to an arrangement of optical components, such as lenses and apertures, that are usually designed to focus light from a scene onto an image plane, with the goal of preserving as much spatial detail as possible. Section 2.1.1.2 and Section 2.1.1.3 introduce simplified camera models for this process, followed by a discussion of common imperfections and aberrations introduced by real-world optical systems in Section 2.1.1.4.

Optical system

2.1.1.2 Pinhole camera model

The *pinhole camera model* is a fundamental and idealized model of image formation. It abstracts the optical system to an infinitesimally small aperture through which all light rays from the scene must pass before reaching the image plane. This means that every object-side ray has a one-to-one correspondence to a point on the image plane, and all objects are perfectly in focus. Despite its simplicity, this model is

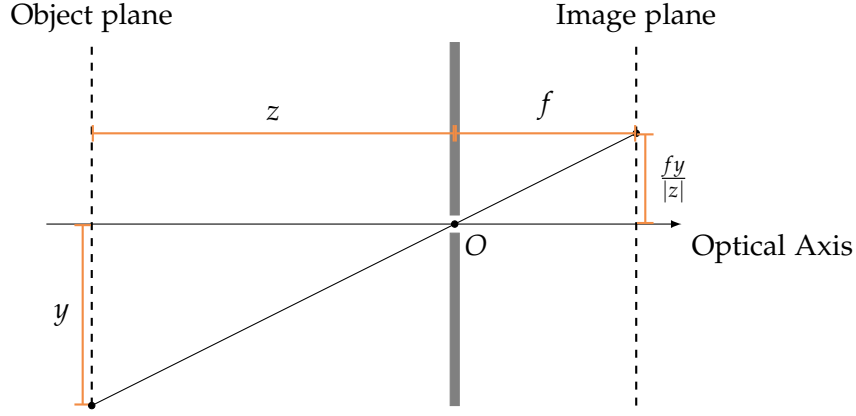


Figure 2.1: Pinhole Camera Model.

used in many algorithms in 3D reconstruction, image analysis, and other computer vision tasks [FP12].

In the pinhole model, the aperture is placed at the origin of a 3D coordinate system (see Figure 2.1). The view direction is along the negative z -axis and is also referred to as the *optical axis*. The pinhole divides the coordinate system into an object side, with negative z coordinates and an image side, with positive z coordinates. The image plane is located behind the pinhole on the image side at a distance equal to the focal length f . Scene points are projected through the pinhole onto this image plane, forming an inverted image. To avoid dealing with image inversion, an equivalent formulation is often used in which a virtual image plane is placed *in front* of the pinhole at a distance f . This convention will also be used in the following explanations.

optical axis

Virtual image

Let a point in the scene be given by $\mathbf{P} = (x, y, z)^\top \in \mathbb{R}^3$. In the ideal pinhole model, this point projects onto an image coordinate $\mathbf{P}' = (u, v)^\top \in \mathbb{R}^2$ on the image plane according to the following equations:

$$u = \frac{fx}{z}, \quad v = \frac{fy}{z} \quad (2.1)$$

This transformation captures the perspective projection effect: objects farther from the camera appear smaller on the image plane. In homogeneous coordinates, this can be written as the matrix equation

Perspective projection

$$z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}. \quad (2.2)$$

In practical imaging systems, the coordinates of points on the image plane are typically measured in *pixel units* relative to one of the sensor corners rather than continuous physical units (e.g. millimeters), and

Pixel units

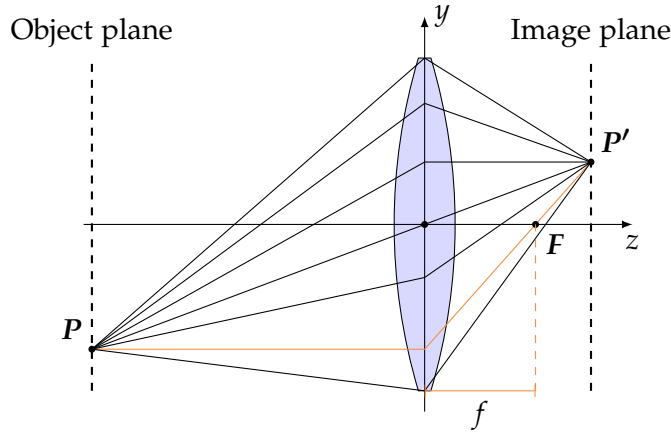


Figure 2.2: Thin lens model with the object plane perfectly in focus. Light rays originating from a point source P in the object plane converge to a single point P' in the image plane. Rays parallel to the optical axis (orange) always pass the focal point F .

it is then convenient to include offset parameters that represent the *principal point*, i.e. the point where the optical axis intersects the image plane. In such cases, the image coordinates in pixel units are given by $u = \frac{f_x x}{z} + c_x$ and $v = \frac{f_y y}{z} + c_y$, or in matrix notation:

Principal point

$$z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (2.3)$$

Here, (c_x, c_y) represents the coordinates of the principal point in the image, and f_x and f_y represent the focal lengths in terms of pixel dimensions in order to allow rectangular pixels.

Although the pinhole camera model is useful for a variety of tasks because of its mathematical simplicity, it lacks the ability to capture several important real-world phenomena, such as optical aberrations. Since everything is in focus, it cannot model depth of field and out-of-focus blur effects. Furthermore, an (even approximately) infinitesimally small aperture is impractical, since it would block almost all incoming light, resulting in underexposed images with a low signal-to-noise ratio (see [Section 2.1.2](#)). The model also ignores wave optical effects, such as diffraction blur, that would become relevant at such small aperture sizes.

Limitations of pinhole model

2.1.1.3 Thin lens model

One way to increase the amount of light reaching the sensor is to increase the size of the aperture, allowing more rays from a given object

point to pass through the optical system. However, this introduces defocus blur: light rays from a single object point no longer converge at a single location on the image plane. Instead, they intersect the image plane at different positions, causing the light to spread over a region rather than forming a sharp point. Conversely, a single point in the image plane may now receive contributions from multiple object points, making it difficult to distinguish them. This overlap results in defocus blur and loss of spatial detail in the resulting image.

Blur

To avoid this, an optical system should ideally focus all light rays originating from an object point and passing the aperture onto a single image point. Figure 2.2 shows the most basic lens model, i.e. the thin-lens model that consists only of a single lens whose thickness is considered negligible compared to the curvature of the spherical lens surfaces [Sze22]. Like in the pinhole model, the lens is centered at the origin of the scene's coordinate system. The plane at the origin perpendicular to the optical axis is called the principal plane. Using the laws of refraction, the lensmaker's equation can be derived as

Lensmaker's equation

$$\frac{1}{f} = (n - 1) \left(\frac{1}{R_1} - \frac{1}{R_2} \right), \quad (2.4)$$

where f is the focal length of the lens, n is the index of refraction of the lens material and R_i are the radii of the curvature of the lens surfaces. The derivation of this equation also makes use of the *paraxial approximation*, i.e., that the angle between rays and the optical axis is small, which is the case when objects are sufficiently far from the camera [Hec17].

Light rays passing through the lens obey the following rules [Sze22]:

1. Rays passing through the center of the lens do not change directions.
2. All object-side rays parallel to the optical axis intersect it at distance f behind the lens. This point is called the focal point F .
3. Rays intersecting the optical axis at distance f on the object side become parallel to the optical axis on the image side.
4. Multiple object side rays that are parallel to each other, but not necessarily parallel to the optical axis, all converge on the plane containing F that is perpendicular to the optical axis. This plane is called the *focal plane*.

Interchanging the terms *object side* and *image side* in these rules also applies, e.g. rays intersecting the optical axis at the distance f on the *object side* become parallel to the optical axis on the image side.

In the following, let d_o denote the absolute distance from the lens to the object point along the optical axis, and d_f the corresponding perfect focus image distance (i.e., where rays originating from the

object point converge). Let d_i be the actual distance from the lens to the image or sensor plane (which may or may not equal d_f). The relationship of d_o and d_f is given by the *Gaussian lens equation* [Hec17, p. 159]:

$$\frac{1}{f} = \frac{1}{d_o} + \frac{1}{d_f}. \quad (2.5)$$

Gaussian lens equation

This means that if the image plane (i.e., the sensor) is positioned at distance d_f behind the lens, all points in the object plane at distance d_o in front of the lens are in focus. Rearranging [equation \(2.5\)](#) for d_f yields the convergence distance of the rays as a function of d_o :

$$d_f = \frac{fd_o}{d_o - f}, \quad (2.6)$$

which is also where the image plane needs to be placed to focus at distance d_o . The image coordinates (u, v) in the image plane can then be computed analogously to the pinhole model, using the factor $M = \frac{f}{f - d_o} = -\frac{d_f}{d_o}$:

$$u = Mx, \quad v = My. \quad (2.7)$$

The magnitude $|M|$ is called *magnification* and represents the ratio of the object size on the object and the image plane. For a real image, the object distance satisfies $f < d_o$. Since $d_o > 0$ and $d_f > 0$ denote absolute distances, the factor M itself is negative, resulting in an inverted real image.

Magnification

As the object distance $d_o \rightarrow \infty$, the image distance d_f approaches f in [equation \(2.5\)](#). This means that to focus on distant objects, the sensor must be placed at the focal length behind the lens, which analogous to the configuration in the pinhole camera model [Sze22].

Focus at infinity

If a point P is not in the object plane, i.e. the plane the camera is currently focused on, the light rays intersect the image plane at multiple points, which again results in blur. The radius of the blur circle formed on the image plane by a point P that is not in focus is called *Circle of Confusion (CoC)*. The *CoC* provides a quantitative measure of defocus and is useful for defining whether a point is considered acceptably sharp. Let A be the diameter of the aperture, d_f the distance where the rays from P converge and d_i the actual distance of the image plane to the principal axis. The *CoC* can then be computed as [PC82]

Circle of Confusion

$$c = A \frac{|d_f - d_i|}{d_f}. \quad (2.8)$$

It is clear that for $d_f = d_i$ the *CoC* becomes zero, since the point is now in focus.

Unless the scene only consists of a plane orthogonal to the optical axis, not all objects can be focused at once, since the objects are at

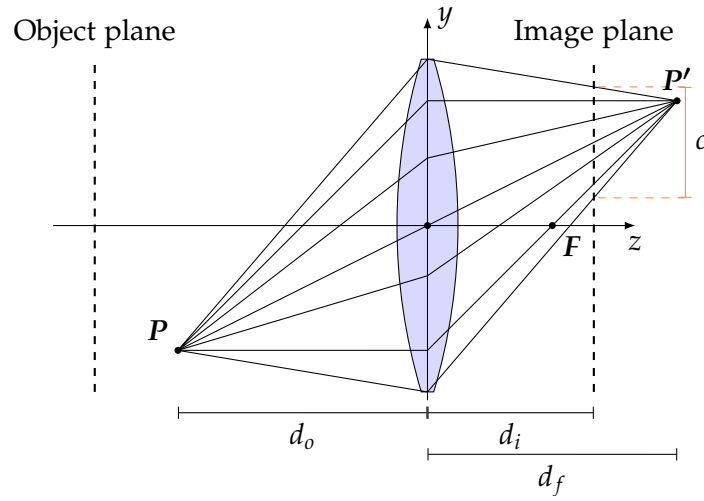


Figure 2.3: Imaging a point out of focus. Light rays originating from the point source P converge behind the image plane. The area in the image plane illuminated by the point source is called the circle of confusion and has diameter c (see [equation \(2.8\)](#)).

different distances. An important concept is the depth of field (DOF), which describes the range in which objects are *acceptably sharp*, i. e. their CoC is smaller than some threshold $c_{\max}$. For digital cameras, this threshold could, for example, be set to be smaller than the size of a single pixel. An approximation of the DOF is usually given as [AT11]

Depth of Field

$$D = \frac{2c_{\max} N d_o^2}{f^2}, \quad (2.9)$$

where d_o is the focus distance and $N = f / A$ is the ratio of focal length and aperture diameter, called *f-number*, which is typically displayed on camera lenses. Smaller f-numbers (larger apertures) thus result in a larger depth of field. Focusing at the *hyperfocal distance* H given by

Hyperfocal distance

$$H = \frac{f^2}{N c_{\max}} + f \quad (2.10)$$

extends the DOF to its maximum from $H/2$ up to infinity [AT11]. The hyperfocal distance is the closest focus distance at which points at infinity are acceptably sharp. Focusing beyond it therefore does not bring distant objects into better focus, but instead only shifts the near limit of the depth of field farther away.

2.1.1.4 Aberrations

Even when a spherical lens is manufactured with perfect precision, optical aberrations inevitably occur. This is because of the geometric assumptions built into the simplified thin lens model.

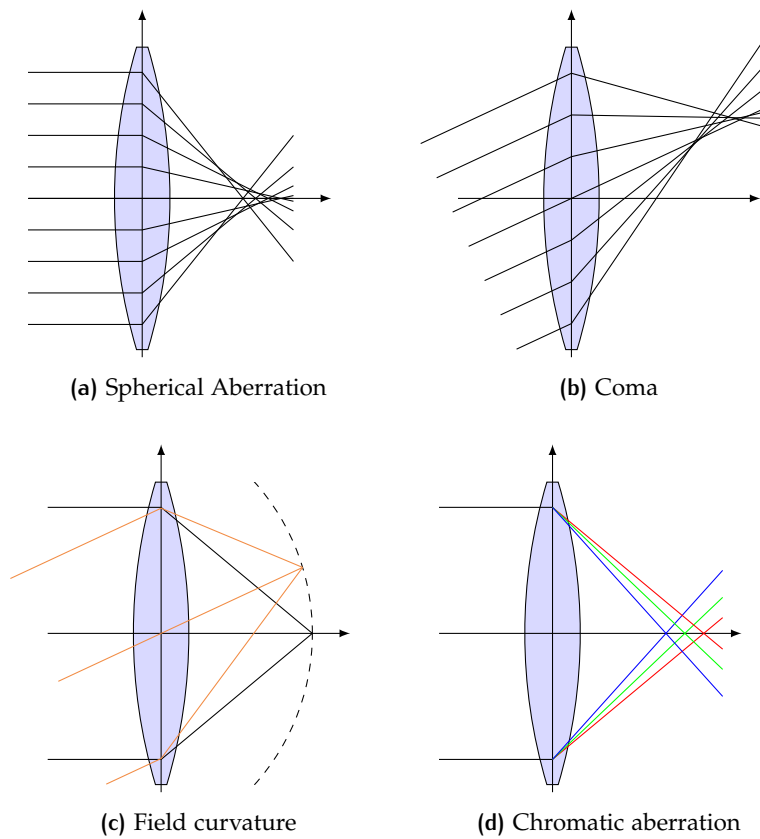


Figure 2.4: Optical aberrations. (a) Light rays parallel to the optical axis no longer converge in the focal point and cause a blurred image. (b) Parallel rays that hit the lens at an angle pass the focal plane in different positions. (c) Parallel rays focus on a curved surface instead of a plane. (d) Light consisting of different wavelengths has multiple focal planes.

The thin lens model assumes that the lens has an infinitesimally small thickness and, consequently, that only a single refraction occurs at the idealized lens plane, rather than separate refractions at the two lens surfaces. Moreover, it relies on the first-order *paraxial approximation*, which assumes that only light rays at small angles with the optical axis reach the lens. This assumption simplifies the lens equations, but breaks down for marginal rays that enter the lens at larger angles.

Paraxial approximation

The deviations that occur when moving from the paraxial approximation to a more accurate third-order approximation are known as *Seidel aberrations* [Hec17, p. 259]. These aberrations are typically classified into the five categories of spherical aberration, coma, astigmatism, distortion, and field curvature. Additionally, there are *chromatic aberrations*, which arise due to the wavelength dependence of the refractive index of a lens. The aberrations are described in more detail below (for further reading, see [Hec17, Ch. 6.3]).

Seidel aberrations

Chromatic aberrations

SPHERICAL ABERRATION On a spherical surface, marginal rays do not refract to the point predicted by the thin lens model. This means that rays parallel to the optical axis on the object side will not converge exactly at a single point after passing the lens (see Figure 2.4(a)). This phenomenon is known as *spherical aberration*. The discrepancy between the focal points of the paraxial and marginal rays leads to a blurred image.

COMA For off-axis object points, parallel rays that enter the lens at an angle to the optical axis do not converge at a single point on the focal plane. This results in a blur that, unlike spherical aberration, has an asymmetrical, comet-like intensity distribution, which is why this aberration is called *coma* (see Figure 2.4(b)).

ASTIGMATISM *Astigmatism* occurs for off-axis object points, where rays lying in two orthogonal planes focus at different distances. If the camera is focused such that the rays in one of those planes converge on the image plane, the rays in the other plane will form an elongated blur spot.

DISTORTION The magnification factor M is typically not constant, but depends on the distance from the optical axis. This causes objects near the image corners to either appear larger (*pincushion distortion*) or smaller (*barrel distortion*) than objects near the center. This aberration does not cause blur on its own, and can often be removed by software after the image is taken with the help of camera calibration models, e.g., [Zha00].

FIELD CURVATURE The ideal thin lens model assumes a flat image plane, but spherical lenses actually focus incoming rays onto a curved surface. When the sensor is flat, only a portion of this

curved surface is in perfect focus, causing regions toward the edges to appear blurred (see [Figure 2.4\(c\)](#)).

CHROMATIC ABERRATION Because the refractive index of a lens material varies slightly with wavelength, different colors are refracted by different amounts. This means light of varying wavelengths converges at different points, resulting in colored fringes around edges in the image (see [Figure 2.4\(d\)](#)).

In real optical systems, these aberrations occur in combination rather than in isolation. High-end real-world camera lenses often mitigate these aberrations with more complex optical systems, consisting of multiple lens elements. The principle is that aberrations from one element can be partially canceled by carefully designed opposing aberrations from another element. The description of multi-lens systems is more complex than the models presented in this dissertation. A comprehensive overview can again be found in [\[Hec17\]](#). Despite these mitigation techniques, simpler lens systems with inherent aberrations remain prevalent in many computer vision applications due to cost, weight, and size constraints, making computational deblurring and aberration correction techniques still highly relevant.

2.1.2 Measurement

The previous section described how optical systems are designed to reshape light rays originating from a scene to ideally focus on an image plane. This section describes how the light hitting the sensor is measured and converted into a digital representation, specifically in [CMOS](#) sensors.

2.1.2.1 Radiometric quantities

Light, as electromagnetic radiation, can be interpreted as a stream of individual photons. Each photon hitting a sensor pixel has a chance to generate an electric charge, that is accumulated, converted to a voltage and then into a digital representation. The total charge accumulated in a pixel is proportional to the number of photons and thus the intensity of light hitting the pixel. To understand the relationship of light rays in the context of geometric optics in the last section and this high-level overview, basic radiometric quantities need to be introduced [\[PJH23\]](#). This connection between geometric optics and sensor measurement is especially useful for sensor simulation, e. g. with raytracing algorithms.

An individual photon has an energy depending on its wavelength

$$E(\lambda) = \frac{hc}{\lambda}, \quad (2.11)$$

where h is the Planck constant, c is the speed of light in a vacuum, and λ is the wavelength of the photon. The total amount of *radiant*

energy Q_e (measured in joules J) of the stream of photons is the sum of the individual energies, which for a continuous wavelength band turns into an integral

Radiant energy

$$Q_e = \int_0^\infty N(\lambda)E(\lambda) d\lambda, \quad (2.12)$$

where $N(\lambda)$ is a density function that describes the number of photons with a wavelength in a small interval around λ^1 .

While radiant energy Q_e describes the total energy of electromagnetic radiation, imaging systems are often more concerned with how much energy is received in a short period of time. *Radiant flux* Φ is defined as the radiant energy per unit time

Radiant flux

$$\Phi_e = \frac{dQ_e}{dt} \quad (2.13)$$

measured in watts ($W = J/s$). This quantity describes the rate at which radiant energy is emitted or received. In the context of image sensors, the total electromagnetic energy transferred to a pixel is the radiant flux integrated over the *exposure time*.

Exposure time

The individual pixels of an image sensor only have a finite area. Intuitively, the number of photons that hit a pixel is influenced by the size of that area. The flux per unit area is called *irradiance* and is defined as

Irradiance

$$E_e = \frac{d\Phi_e}{dA}. \quad (2.14)$$

Integrating irradiance over the pixel area A yields radiant flux, and integrating irradiance over the exposure time yields *radiant exposure*

Radiant exposure

$$H_e = \frac{dQ_e}{dA} = \int_{t_{\min}}^{t_{\max}} E_e(t) dt. \quad (2.15)$$

Every point inside a pixel can be hit by light rays from all directions on a hemisphere. *Radiance* is the flux per unit area and per unit steradian and is defined as

Radiance

$$L_e = \frac{d^2\Phi_e}{d\omega dA \cos\theta'}, \quad (2.16)$$

where ω is a solid angle and θ the angle between the normal of the surface and the incident light ray. Radiance can be seen as the flux contribution of a single ray of light and is thus fundamental in many algorithms in computer graphics, such as path tracing [PJH23].

There are spectral versions for each radiometric quantity that describe the contribution of each wavelength to the total ,e. g., *spectral radiance* is defined as

$$L_{e,\lambda} = \frac{dL_e}{d\lambda}, \quad (2.17)$$

¹ In the literature, the subscript e usually distinguishes radiometric from photometric quantities, which then have subscript v . This convention will also be used throughout this thesis.

which can be converted back by integrating over all wavelengths, i. e.,

$$L_e = \int_0^\infty L_{e,\lambda}(\lambda) d\lambda. \quad (2.18)$$

Given a spectral radiance function $L_{e,\lambda}(\mathbf{p}, \boldsymbol{\omega}, t, \lambda)$, that computes the radiance at every position $\mathbf{p}$ from direction $\boldsymbol{\omega}$ at time t , the radiant energy arriving at a single pixel k can now be computed as

$$Q_e^k = \int_{\lambda_{\min}}^{\lambda_{\max}} \int_{t_{\min}}^{t_{\max}} \int_{A_k} \int_{\Omega} L_{e,\lambda}(\mathbf{p}, \boldsymbol{\omega}, t, \lambda) \cos(\theta) d\boldsymbol{\omega} d\mathbf{p} dt d\lambda. \quad (2.19)$$

The integral computes the radiant energy by taking the contributions of all rays of all wavelengths over the exposure time $[t_{\min}, t_{\max}]$, coming from the hemisphere of all possible directions Ω , and hitting the set of all points inside the pixel's area A_k . Note that the aperture of the optical system in front of the sensor actually restricts the set of directions that actually have contributions to a subset of the hemisphere.

2.1.2.2 Photon to charge conversion

A typical camera sensor consists of an array of photodiodes that are capable of converting light into electric currents. Each photon hitting a photodiode has the probability $\eta(\lambda)$, known as *quantum efficiency* to generate an electric charge. Thus, for *monochromatic light* consisting of N photons of a single wavelength, the actual measured charge q is Poisson distributed with expected value $\bar{q}$ proportional to the intensity of incident light and is expressed as

Quantum efficiency

$$\bar{q} = \eta(\lambda)eN = \eta(\lambda)e \frac{Q_e}{E(\lambda)} = \eta(\lambda)e \frac{\lambda}{hc} Q_e, \quad (2.20)$$

where e is the elementary charge of an electron. The probabilistic nature of charge generation is an unavoidable error source that will be further analyzed in [Section 2.1.2.5](#).

The factor

$$R(\lambda) = \eta(\lambda)e \frac{\lambda}{hc} \quad (2.21)$$

in [equation \(2.20\)](#) is called *responsivity* and gives the conversion ratio of optical energy to electric current of the sensor (in A/W) [[Hui20](#)]. The responsivity of silicon, which is the most common semiconductor used in [CMOS](#) sensors, grows nearly linearly with wavelength in the visible wavelength domain, but drops off sharply at around 1000 nm (see [Figure 2.5](#)).

Responsivity

Absorbing the responsivity factor into the definition of Q_e^k inside [equation \(2.20\)](#), and integrating the contributions over all wavelengths by inserting [equation \(2.19\)](#) gives the expected charge at pixel k [[HK94](#)]

$$\bar{q}_k = \int_{\lambda_{\min}}^{\lambda_{\max}} \int_{t_{\min}}^{t_{\max}} \int_{A_k} \int_{\Omega} R(\lambda) L_{e,\lambda}(\mathbf{p}, \boldsymbol{\omega}, t, \lambda) \cos(\theta) d\boldsymbol{\omega} d\mathbf{p} dt d\lambda. \quad (2.22)$$

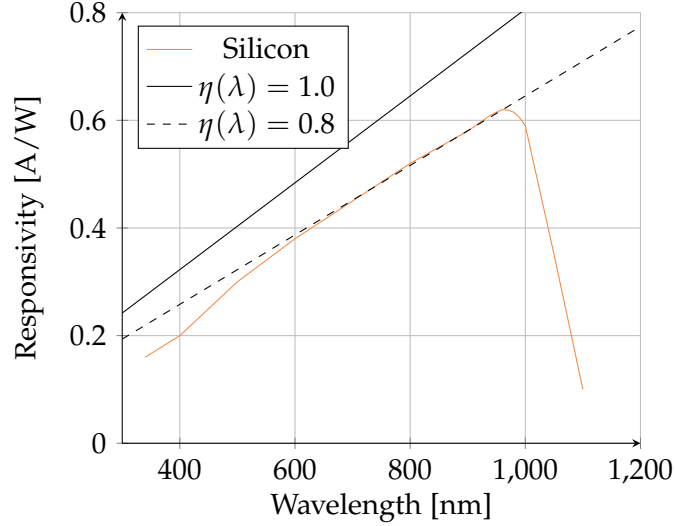


Figure 2.5: Responsivity curve of a typical silicon photodiode and reference lines for constant quantum efficiencies. Data taken from the Thorlabs FDS10X10 data sheet [14].

Note that it is assumed here that the responsivity is constant over time and all positions inside the pixel. The collected charge is converted to a voltage, amplified and further converted into digital numbers. In an idealized sensor model [EMV21], this processing is linear, such that the final digital number

$$\bar{I}_k = g\bar{q}_k \quad (2.23)$$

with linear gain g is proportional to the energy of the incident light. Note, that a real sensor would also include a quantization step which is ignored for now for simplicity.

In real cameras, the pixel size $|A_k|$ and often also the exposure time $\Delta t = t_{\max} - t_{\min}$ are very small. If the radiance inside of a pixel and over the exposure is seen as approximately constant, the collected charge and thus $\bar{I}_k$ are up to first order also proportional to the radiant exposure (because $H_e = \frac{dQ_e}{dA}$ means $Q_e \approx H_e|A_k|$) or the irradiance hitting that pixel (because $E_e = \frac{d^2Q_e}{dt dA}$ means $Q_e \approx E_e\Delta t|A_k|$), so after normalizing by exposure time and/or pixel size, cameras can also be seen as radiant exposure or irradiance measurement devices. In the subsequent chapters of this dissertation, these proportionality constants (including the gain g) and the actual physical quantity of the digital numbers that the sensor measures are not important, and $\bar{I}_k$ will simply be referred to as *intensity*, which is proportional to the incident light.

Interpretation of $\bar{I}_k$

2.1.2.3 Color filters

The measurement described by equation (2.22) integrates the contributions of light over all wavelengths, resulting in a single intensity value

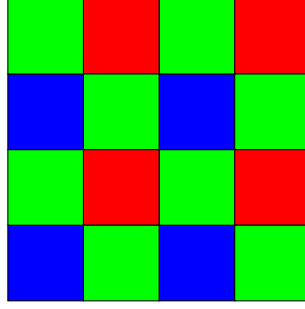


Figure 2.6: Bayer pattern [Bay76]

for each pixel. Although this approach works well for monochrome imaging, it loses the spectral information of all imaged objects, which is often required to maximize performance of downstream applications, like accurate color reproduction for human viewers.

In most real-world imaging systems, the sensor itself is not inherently color sensitive. Instead, color information is obtained by applying a color filter array (CFA) in front of the sensor, which is arranged in a specific pattern. A common example is the Bayer pattern [Bay76] in Figure 2.6, whose filters are arranged such that twice as many pixels measure green light compared to red and blue light. This CFA pattern is optimized for human perception, since color receptors in the eye are the most sensitive to colors in the green spectrum.

Color filter array

Each color filter is characterized by a wavelength-dependent transmission function $S_{c_k}(\lambda)$, where the subscript c_k indicates the color channel of pixel k (e.g., red, green, or blue for the Bayer pattern). The role of $S_{c_k}(\lambda)$ is to modulate the intrinsic responsivity $R(\lambda)$ of the sensor, so that only wavelengths passing through the filter contribute to the charge accumulation in that pixel. The expected charge with color filters is thus given by

$$\bar{q}_k = \int_{\lambda_{\min}}^{\lambda_{\max}} \int_{t_{\min}}^{t_{\max}} \int_{A_k} \int_{\Omega} R(\lambda) S_{c_k}(\lambda) L_{e,\lambda}(\mathbf{p}, \boldsymbol{\omega}, t, \lambda) \cos(\theta) d\boldsymbol{\omega} d\mathbf{p} dt d\lambda. \quad (2.24)$$

Notably, the transmission functions do not only select a single wavelength (i.e., they are not Dirac delta functions), but consist of potentially overlapping curves. After the sensor has captured the raw data through the CFA, each pixel only contains information for a single color channel. To reconstruct a full color image, the missing information needs to be approximated by *demosaicing* algorithms as part of the ISP pipeline.

Demosaicing

2.1.2.4 Measurement equation

In addition to color filters, many other factors modulate the sensor response, e.g., photo response non-uniformity (PRNU), dark signal

non-uniformity (DSNU), per-pixel gain, micro-lens arrays and spectral filtering by the optical system itself. Instead of modeling each separately, all these effects can be combined in a single weighting function $W_k(p, \omega, t, \lambda)$, which includes the CFA transmission $S_c(\lambda)$, sensor responsivity $R(\lambda)$, the cosine attenuation term $\cos(\theta)$ and also the (ideally) linear conversion factor from charge to digital numbers. The generalized measurement equation [PJH23; Nim+20] for the expected value of an intensity measurement $\bar{I}_k$ at pixel k becomes

Measurement equation

$$\bar{I}_k = \int_{\lambda_{\min}}^{\lambda_{\max}} \int_{t_{\min}}^{t_{\max}} \int_{A_k} \int_{\Omega} W_k(\mathbf{p}, \omega, t, \lambda) L_{e,\lambda}(\mathbf{p}, \omega, t, \lambda) d\omega d\mathbf{p} dt d\lambda. \quad (2.25)$$

This is the most general form of the measurement equation, capturing all spatial, angular, temporal and spectral dependencies of a sensor inside W_k . Although not specified for this general formulation, the methods developed in this thesis often assume that W_k is at least approximately separable in some of its arguments and can be written as a product $W_k(\mathbf{p}, \omega, t, \lambda) = W_{p,k}(\mathbf{p})W_{\omega,k}(\omega)W_{t,k}(t)W_{\lambda,k}(\lambda)$. Equation (2.25) will become the basis for optimizing the sensor parameters in Chapters 3 and 4, where W_k will also depend on the learnable parameters.

2.1.2.5 Error sources

The measurement process is subject to several error sources that influence the quality of the captured image signal. While some of these errors arise from imperfections in the sensor circuitry, there are also error sources due to fundamental quantum mechanical processes that cannot be eliminated.

FIXED PATTERN NOISE Due to manufacturing differences, not all pixels in a sensor array behave in the same way [HK94]. The variability in responsivity between pixels leads to photo response non-uniformity (PRNU) and can be measured by capturing a uniform light source. In addition, a signal could be measured even for a completely dark scene, which is called dark signal non-uniformity (DSNU). These non-uniformities do not change over time, but their strength can depend on other factors, like the temperature of the sensor or exposure time. Since these error sources are not random and effectively consist of a per-pixel gain and offset, they can be estimated by capturing calibration images, e.g., a dark frame for the DSNU and a uniformly illuminated (white) image for the PRNU. The measured signal can then be corrected by applying a per-pixel linear transformation.

TEMPORAL NOISE As described in Section 2.1.2.2, the conversion of photons to electrons within an image sensor is an inherently random process. Photons of wavelength λ are only converted

with the probability given by the quantum efficiency $\eta(\lambda)$. The noise from this random process is called *shot noise*, and follows a Poisson distribution [HK94], i. e., for monochromatic light the probability to generate k electrons from N photons is

Shot noise

$$P(k) = \frac{\bar{n}_e^k}{k!} \exp(-\bar{n}_e), \quad (2.26)$$

where the scale parameter $\bar{n}_e = \eta N$ is the expected number of electrons. Since the sum of Poisson distributed random variables is again Poisson distributed, electrons generated from polychromatic light also follow a Poisson distribution. Since mean and variance of the Poisson distribution are the same, and the conversion from charge to intensity measurement is linear, the signal-to-noise ratio (SNR) of the final image signal (see equation (2.23)) is given by [HK94]

$$\text{SNR} = \frac{\bar{I}}{\sqrt{\bar{I}}} = \frac{g\bar{q}_k}{\sqrt{g^2\bar{q}_k}} = \frac{\bar{q}_k}{\sqrt{\bar{q}_k}} = \sqrt{\bar{q}_k} \quad (2.27)$$

This means that the SNR becomes better for a higher number of photons, which can be achieved through various means, e. g., larger sensor area, larger aperture, higher exposure time or color filters with wider spectra. The dependence of the SNR on pixel sizes and the shape of color filters becomes crucial in Chapters 3 and 4.

DISCRETIZATION ERRORS Since there is only a finite number of pixels in a sensor array, the lateral resolution of images is necessarily limited, and small details in the scene might not be captured. Similarly, the quantization of the signal by an analog-to-digital converter introduces *quantization errors* which can be modeled as uniformly distributed noise between two quantization steps.

MOTION BLUR This error arises due to movements in the scene during the exposure time, resulting in objects being blurred in the direction of movement.

There are trade-offs between the various error sources. For instance, even though a shorter exposure time decreases motion blur and smaller pixels improve lateral resolution, they both worsen the SNR due to shot noise, since less light is captured per pixel.

2.1.2.6 Time-of-Flight sensors

The objective of *range imaging* is to measure the distance between the camera and scene surfaces at every pixel of an image grid. Besides triangulation-based methods like stereo imaging, which estimate distances from disparity between two images captured by cameras separated by a known baseline, there are active methods such as time

of flight (ToF) sensors that determine distance directly by measuring the round-trip time of light emitted by a near-infrared (NIR) source colocated with the camera.

There are several ways to implement a ToF sensor. A popular choice is amplitude-modulated continuous wave (AMCW) ToF [Lanoo; Kol+10] in photonic mixing device (PMD) sensors, where the illumination intensity is modulated with a known sinusoidal reference signal $g(t) = \cos(2\pi f_m t)$ with known modulation frequency f_m . This leads to a phase-shifted optical signal modeled as $s(t) = b + a \cos(2\pi f_m t + \phi)$ at the sensor, where a and b are amplitude and bias terms determined by surface properties, background illumination and absolute intensity of the light source, and ϕ is the phase shift dependent on the distance d to the sensor.

AMCW

The time delays are determined by repeated measurement of the irradiance over multiple exposure windows [Sch11]. The physical principles (i. e. photon-to-charge-conversion) of the capturing process are the same as color sensors described in the previous sections. In a PMD pixel, the photocharges generated in the optical sensitive areas are routed into two "buckets" according to an electrical field modulated by the reference signal $g(t)$. The difference between the accumulated charges in these buckets implements a correlation measurement between the received optical signal and the reference. The correlation at time offset τ is [Lanoo; Kol+10]

$$C(\tau) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{T/2} s(t)g(t + \tau) dt \quad (2.28)$$

$$= \frac{a}{2} \cos(2\pi f_m \tau + \phi) + b. \quad (2.29)$$

Although in practice the exposure time T is finite, it is usually much longer than a modulation period, which makes the approximation of the limit accurate. Repeating the measurement at four different phase offsets of the reference yields four correlation measurements, called phase images, $A_i = C(i \cdot \frac{\pi}{2})$, from which the phase shift ϕ can be computed as

$$\phi = \arctan2(A_3 - A_1, A_0 - A_2). \quad (2.30)$$

Finally, with round-trip time $2d/c$, the phase and the distance are related by

$$d = \frac{c}{4\pi f_m} \phi, \quad (2.31)$$

where c is the speed of light.

Compared to conventional image sensors, depth estimates from ToF sensors are affected by several unique error sources, which are complicated by the non-linear dependence of the final depth image on the individual phase images.

Error sources

LATERAL RESOLUTION Although the resolution of consumer grade ToF sensors improved over the years, it still remains relatively low compared to conventional image sensors. This is due to the use of larger pixel areas because of the space required for the more complex pixel electronics, and to reducing noise by capturing more light [Sch11; NIL12].

LIMITED RANGE Since $\phi \in [0, 2\pi)$, the unambiguous range is naturally limited to $d_{\max} = \frac{c}{2f_m}$, after which depth aliasing occurs. There are ways like multi-frequency schemes [GYBo4] that can extend this range.

MULTIPATH INTERFERENCE Light rays emitted by the NIR source are not necessarily reflected directly back to the camera, but can have multiple bounces, leading to incorrect depth estimates, e. g., rounded corners [BLK17; Sch11]. This effect can also arise at translucent surfaces in the scene.

STATISTICAL NOISE Like conventional cameras, ToF cameras suffer from shot noise. The SNR also depends on the NIR reflectivity of the scene, where the noise is relatively stronger in darker areas. Additionally, the intensity of the active point light source decreases the further the light has to travel. This leads to spatially varying axial and lateral noise whose strength increases with distance [NIL12].

FLYING PIXELS A pixel footprint overlapping depth discontinuities, e. g., at object boundaries, leads to a computed depth value that does not belong to any of the involved surface elements, and can even be in front of the foreground or behind the background surface because of the non-linear nature of the depth reconstruction (equation (2.30)). This results in large outlier measurements, which appear to be “flying” over the actual surfaces, thus commonly referred to as *flying pixels* [Sch11].

TEMPORAL ARTIFACTS Because phase images are taken at successive time steps, changes in background illumination and motion in either the scene or the camera itself leads to artifacts. Due to the non-linearity in equation (2.30), these artifacts are different than simple blurring of the depth values. Motion compensation schemes, specifically for phase images of ToF imagers, have been proposed [Ste+19a; Sch11; HLK13].

SYSTEMATIC ERRORS Imperfections in the emitted modulation signals, which in practice deviate from ideal sinusoids, introduce a systematic error called *wiggling* [Kol+10].

2.2 DEEP LEARNING

Over the last decade, deep learning has achieved remarkable success in a wide range of fields, including natural language processing, computer vision, computer graphics and robotics. This section introduces the fundamental concepts of supervised deep learning and neural networks, and outlines how these models are integrated within modern imaging pipelines.

At its core, a neural network is a highly flexible, data-driven model capable of approximating complex functions. This versatility makes neural networks applicable to a large range of image processing tasks, such as denoising, demosaicing, classification, semantic segmentation, and image generation. [Section 2.2.1](#) explains the foundational aspects of neural network training and the overall objective.

The term *deep learning* refers to neural networks that consist of many *layers*, which are building blocks that are highly composable and, when stacked, allow for the capturing of complex hierarchical patterns in the data. The most common layer types for computer vision tasks are explained in [Section 2.2.2](#).

Every deep learning application relies on a well-defined objective function, or *loss*, which is minimized during the training process. [Section 2.2.3](#) introduces the most commonly used loss functions, while [Section 2.2.4](#) explains the optimization algorithms that adjust the network parameters during training.

2.2.1 Overview

The goal of supervised machine learning, in general terms, is to approximate functions based on data. Given training data stemming from some unknown distribution $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \sim p_{\text{data}}$, where $\mathbf{x}^{(i)}$ are inputs and $\mathbf{y}^{(i)}$ are the desired outputs² determined by some unknown function g , i. e., $g(\mathbf{x}^{(i)}) = \mathbf{y}^{(i)}$, the goal is to find a set of parameters θ such that a parameterized function $\mathcal{G}(\cdot; \theta)$ closely approximates g :

$$\mathcal{G}(\mathbf{x}^{(i)}; \theta) \approx g(\mathbf{x}^{(i)}) = \mathbf{y}^{(i)}, \quad (2.32)$$

while also generalizing well to unseen test data that was not included in the training set.

The training objective is stated as finding the parameters θ^* that minimize the risk

$$\theta^* = \operatorname{argmin}_{\theta} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim p_{\text{data}}} (\mathcal{L}(\mathcal{G}(\mathbf{x}; \theta), \mathbf{y})). \quad (2.33)$$

The *loss function* $\mathcal{L}$ quantifies how well the output of the function with

Loss function

² In this section, superscripts are used to indicate the index in the training or test set and subscripts are used for individual components, e. g., $x_j^{(i)}$ refers to component j of the i -th training example.

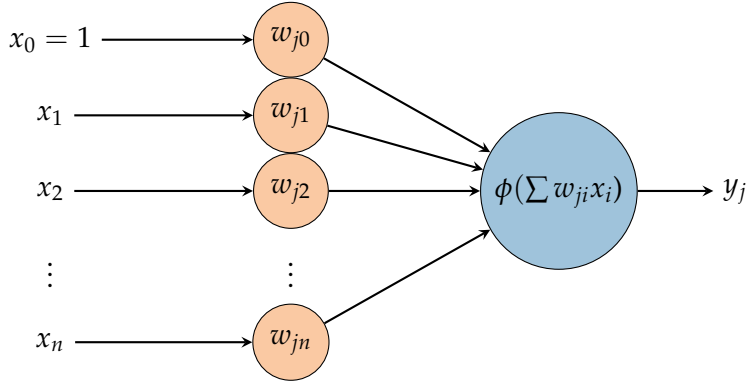


Figure 2.7: Artificial neuron. The learnable weights (orange) are multiplied by the inputs. The result is summed and passed through an activation function ϕ .

the given θ fits the desired output y , and the concrete choice of the loss function depends on the task to be solved. Since the distribution p_{data} is unknown, the expectation value cannot be computed directly, so in practice the *empirical risk* over the finite training set $(x^{(i)}, y^{(i)}), 1 \leq i \leq n$ is minimized instead

Empirical risk minimization

$$\theta^* = \operatorname{argmin}_{\theta} \frac{1}{n} \sum_i \mathcal{L}(\mathcal{G}(x^{(i)}; \theta), y^{(i)}). \quad (2.34)$$

Although the above explanations, in principle, apply to any parameterized function, neural networks have emerged as a particularly powerful class of function approximators. Inspired by biological neurons, these networks are composed of interconnected *artificial neurons*, as depicted in Figure 2.7. The output of a single neuron with weight vector $w_j \in \mathbb{R}^n$, learnable bias $b_j \in \mathbb{R}$ and input vector $x \in \mathbb{R}^n$ can be compactly described with

Neural networks

$$y_j = \phi(\langle w_j, x \rangle + b_j), \quad (2.35)$$

where ϕ is a non-linear activation function (further discussed in Section 2.2.2). The output of one neuron is a scalar, and combining the outputs of multiple neurons forms a *layer*. One of the simplest types of layers is the *fully connected layer*, in which every input is connected to every neuron. A fully connected layer (illustrated Figure 2.8) can be described by stacking the weights of each neuron in a matrix $W \in \mathbb{R}^{m \times n}$ and the biases into a vector $b \in \mathbb{R}^m$

Layer

$$y = \phi(Wx + b) \in \mathbb{R}^m. \quad (2.36)$$

Multiple layers $f_l(\cdot; \theta_l)$ can be stacked to form the complete network

$$\mathcal{G}(x; \theta) = f_L(f_{L-1}(\dots f_1(x; \theta_1); \dots); \theta_{L-1}); \theta_L), \quad (2.37)$$

where θ_l refers to a vector containing all learnable parameters of that layer, e. g., in the case of fully connected layers, all entries of W and

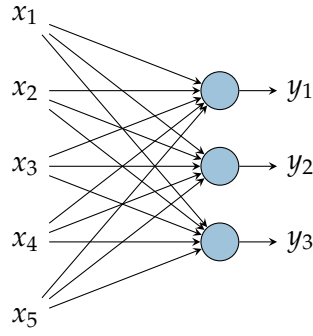


Figure 2.8: Fully connected layer with 5 inputs and 3 outputs (bias inputs are omitted). Each input is fed into each neuron. Every neuron has their own set of weights (see [Figure 2.7](#)).

b. The intuition for deep networks is that each subsequent layer is able to recognize more abstract and complex patterns based on the features extracted by preceding layers. A network only consisting of fully-connected layers is called multilayer perceptron (MLP). There are many other layer types that are used as building blocks in modern neural networks and will be discussed in the next section.

2.2.2 Layers and other building blocks

This section will introduce the most common layers used in computer vision tasks. There is no consistent definition of the term *layer* in the literature and deep learning frameworks, e.g. sometimes activation function are counted as layers themselves, while other times they are only a part of another layer. Here, a *layer* is defined as a general building block of a neural network that transforms an input into an output, potentially with learned parameters. In the deep learning literature, these inputs and outputs are commonly referred to as *tensors* and can be expressed as multidimensional arrays $x \in \mathbb{R}^{D_1 \times \dots \times D_m}$. The (required) shapes of the input, output and parameter tensors depend on the concrete layer type. For mathematical brevity, the following sections use a special tensor slice notation where capital letters refer to dimensions and indices with the corresponding lower-case letters to a slice along that dimension. For example, for a batch of images $x \in \mathbb{R}^{B \times C \times H \times W}$ with batch size B , channel count C , height H and width W , the notation $x_c \in \mathbb{R}^{B \times H \times W}$ refers to the batch containing only the c -th channel of each image.

2.2.2.1 Activation functions

Activation functions are critical for a neural network to learn to approximate complex functions. Since the composition of linear functions is again linear, without activation functions, a network with multiple

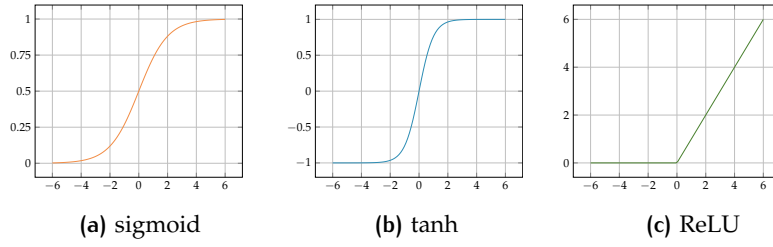


Figure 2.9: Different activation functions.

stacked fully connected layers would only be able to approximate linear functions.

Although in principle any non-linear function can be used as an activation function, the most common ones are sigmoid, Rectified Linear Unit (ReLU) and hyperbolic tangent (tanh), which are applied element wise to a tensor as follows:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \quad (2.38)$$

$$\text{relu}(x) = \max(0, x) \quad (2.39)$$

$$\text{tanh}(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}. \quad (2.40)$$

The three activation functions are shown in [Figure 2.9](#).

Since the output of the sigmoid function falls between 0 and 1, it is often used as the final layer in a binary classification network, because it can be interpreted as the probability that the input belongs to that class. In contrast, the tanh function maps inputs to the range $[-1, 1]$, which can be helpful for restricting the output range in some applications. However, both sigmoid and tanh can suffer from saturation for large inputs, leading to vanishing gradients during training. The ReLU function is computationally efficient and helps mitigate the vanishing gradient problem, making it the most popular choice in modern networks. There are variants of ReLU that set the slope for negative inputs to either a fixed value (LeakyReLU) or making the slope into a learnable parameter (PReLU) [\[Xu+15\]](#). This mitigates the issue of zero gradients for negative inputs.

For multi-class classification, the softmax function is typically used as the final layer. Given an input vector $x \in \mathbb{R}^D$, the softmax function is defined for each output component as

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^D \exp(x_j)}. \quad (2.41)$$

All components of the output of the softmax layer are non-negative and sum to 1, and can thus be interpreted as a probability distribution over the classes.

2.2.2.2 Convolutional layers

Using a fully connected layer with a high-resolution image as input is often impractical because of the large number of parameters required. Each neuron would need a weight for each pixel, leading to high memory and computational requirements. This limitation restricts network depth and overall expressivity, even for images with a moderate number of pixels.

Many computer vision problems are translation invariant, for example, whether an object of a specific class is present in an image or not should not depend on the concrete position of the object on the image plane. Translation invariance can be achieved by making the output of a layer at a specific pixel only depend on a local neighborhood, which is the core idea of *convolutional layers*. For an input $x \in \mathbb{R}^{C_{\text{in}} \times H \times W}$ and learnable kernels $k \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K_h \times K_w}$ the simplest form of the convolutional layer is defined as

$$y_{c_{\text{out}}} = \sum_{c_{\text{in}}=1}^{C_{\text{in}}} x_{c_{\text{in}}} * k_{c_{\text{out}}, c_{\text{in}}}, \quad (2.42)$$

where $*$ is the convolution operation³ applied over the spatial dimensions. This can be interpreted as the convolution with C_{out} different three-dimensional kernels, where the filters do not move in the channel direction.

Since the convolution operator is linear, it can be represented by the multiplication of a matrix with the (vectorized) input image. This matrix (a so-called *Toeplitz* matrix) is sparse and constructed from the kernel weights. As such, a convolutional layer can also be interpreted as a fully connected layer with weight-sharing, which greatly reduces the number of parameters. Like fully connected layers, to break linearity, subsequent convolutional layers require activation functions between them.

There are many extensions to the simple form above, e. g., including a bias term, different padding schemes, a stride greater than 1 to reduce the spatial resolution of the output, or the implementation of a dilated convolution [YK16] to change the receptive field. These extensions provide additional flexibility to solve specific tasks and improve model performance.

2.2.2.3 Pooling layers

Pooling layers are an essential component in many convolutional neural network (CNN) architectures. Their primary purpose is to reduce the spatial dimension of feature maps while retaining the most important information. Similarly to convolutional layers, they work by

³ Deep learning frameworks like PyTorch or TensorFlow actually use cross-correlation instead of convolution, i. e. convolution with a flipped kernel. For convolutional layers this distinction has almost no practical relevance.

aggregating local features in a fixed window that is shifted over the input feature map, however, they do not have learnable parameters but are fixed functions instead. Usually, the stride by which the window shifts is chosen to be greater than 1, which reduces the spatial size of the output. The most commonly used pooling operations are *max pooling* and *average pooling*, which select the maximum value or the average value in a window, respectively.

2.2.2.4 Normalization layers

Normalization layers shift and scale the output of the previous layer to more closely follow a standardized distribution. This stabilizes and accelerates the training, and often provides a regularizing effect that reduces overfitting [Bjo+18]. There are multiple variants of normalization layers, that mainly differ across which dimensions of the input tensors the statistics that determine the scale factors are computed.

Batch normalization is widely used in convolutional networks [IS15]. For an input batch of B training samples $\mathbf{x} \in \mathbb{R}^{B \times C \times H \times W}$, batch normalization normalizes each channel c using the mean and variance over the batch and spatial dimensions:

Batch normalization

$$\mu_c = \frac{1}{BHW} \sum_{b=1}^B \sum_{h=1}^H \sum_{w=1}^W x_{b,c,h,w}, \quad (2.43)$$

$$\sigma_c^2 = \frac{1}{BHW} \sum_{b=1}^B \sum_{h=1}^H \sum_{w=1}^W (x_{b,c,h,w} - \mu_c)^2, \quad (2.44)$$

$$\hat{x}_{b,c,h,w} = \frac{x_{b,c,h,w} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}, \quad (2.45)$$

$$y_{b,c,h,w} = \gamma_c \hat{x}_{b,c,h,w} + \beta_c, \quad (2.46)$$

where $\epsilon > 0$ is a small factor to ensure numerical stability and $\gamma_c, \beta_c \in \mathbb{R}$ are learnable per-channel parameters which determine the mean and scale of the output distribution. The means μ_c and variances σ_c^2 are newly computed for every training batch. Since full batches are not necessarily available during inference, running averages $\hat{\mu}_c$ and $\hat{\sigma}_c^2$ are updated over the training process and replace the batch statistics during inference.

Layer normalization [BKH16] uses statistics independent of the batch size and is standard in transformer architectures (Section 2.2.2.6),

Layer normalization

where it is applied to all feature vectors $x_{b,\cdot} \in \mathbb{R}^D$ of a batch independently:

$$\mu_b = \frac{1}{D} \sum_{d=1}^D x_{b,d}, \quad (2.47)$$

$$\sigma_b^2 = \frac{1}{D} \sum_{d=1}^D (x_{b,d} - \mu_b)^2, \quad (2.48)$$

$$\hat{x}_{b,d} = \frac{x_{b,d} - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} \quad (2.49)$$

$$y_{b,d} = \gamma_d \hat{x}_{b,d} + \beta_d, \quad (2.50)$$

with learnable $\gamma, \beta \in \mathbb{R}^D$.

In **CNNs**, where input tensors have a spatial extent, a common variant is *channel-wise layer normalization* applied per pixel: for $x \in \mathbb{R}^{B \times C \times H \times W}$, normalization is applied for each sample and each pixel position independently, i. e.,

Channel-wise layer normalization

$$\mu_{b,h,w} = \frac{1}{C} \sum_{c=1}^C x_{b,c,h,w}, \quad (2.51)$$

$$\sigma_{b,h,w}^2 = \frac{1}{C} \sum_{c=1}^C (x_{b,c,h,w} - \mu_{b,h,w})^2, \quad (2.52)$$

$$\hat{x}_{b,c,h,w} = \frac{x_{b,c,h,w} - \mu_{b,h,w}}{\sqrt{\sigma_{b,h,w}^2 + \epsilon}}, \quad (2.53)$$

$$y_{b,c,h,w} = \gamma_c \hat{x}_{b,c,h,w} + \beta_c, \quad (2.54)$$

$\gamma, \beta \in \mathbb{R}^C$.

Instance normalization [UVL16] is a variant that was developed for style transfer with the use of **CNNs**. Here, the normalization is done per sample and per channel over the spatial dimensions. For an input $x \in \mathbb{R}^{B \times C \times H \times W}$:

Instance normalization

$$\mu_{b,c} = \frac{1}{HW} \sum_{h=1}^H \sum_{w=1}^W x_{b,c,h,w}, \quad (2.55)$$

$$\sigma_{b,c}^2 = \frac{1}{HW} \sum_{h=1}^H \sum_{w=1}^W (x_{b,c,h,w} - \mu_{b,c})^2, \quad (2.56)$$

$$\hat{x}_{b,c,h,w} = \frac{x_{b,c,h,w} - \mu_{b,c}}{\sqrt{\sigma_{b,c}^2 + \epsilon}}, \quad (2.57)$$

$$y_{b,c,h,w} = \gamma_c \hat{x}_{b,c,h,w} + \beta_c, \quad (2.58)$$

$\gamma, \beta \in \mathbb{R}^C$. Unlike batch normalization, layer normalization and instance normalization do not require running statistics and behave the same during training and inference.

2.2.2.5 Residual connections

Very deep networks suffer from vanishing or exploding gradients, making early layers hard to optimize. *Residual blocks* [He+16] alleviate this issue by creating shortcuts between layers. Usually simple addition is used, i. e. if f denotes a layer (or even a block containing multiple-layers) with parameters θ a residual block is defined by

$$F(x, \theta) = x + f(x; \theta) \quad (2.59)$$

This formulation greatly improves gradient flow (see [Section 2.2.4.2](#)), enabling the training of much deeper architectures than would be possible without residual connections.

When the input and output shapes of a block differ, e. g., the amount of channels changes between convolutional layers or spatial downsampling was applied, a projection shortcut can be used instead

$$F(x; \theta) = P(x) + f(x; \theta), \quad (2.60)$$

where P is a (possibly learnable) linear mapping, e. g., a convolution with kernel size 1 to change the number of channels.

2.2.2.6 Attention and transformer layers

Attention is a mechanism that originated in the field of natural language processing, but has since become a core component in modern image processing architectures as well. In particular, the vision transformer (ViT) architecture [Dos+20] uses an attention variant known as self-attention to capture global dependencies in images. This section explains this self-attention mechanism and its use in the ViT architecture.

At a high level, an attention layer enables the model to transform a sequence of features into another sequence of features, weighted based on the importance of different parts of the input.

An input feature sequence $x \in \mathbb{R}^{n \times d}$ is first projected into different representations, called queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$ and values $V \in \mathbb{R}^{n \times d_v}$, using learnable weight matrices of suitable dimensions

$$Q = W_Q x, K = W_K x, V = W_V x. \quad (2.61)$$

The attention scores are then obtained by computing scaled dot-products between queries and keys, and the values are weighted based on these scores

$$\text{attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V. \quad (2.62)$$

The scaling factor $\sqrt{d_k}$ prevents the input of the softmax function to grow too large, which ensures more stable gradients during training.

A common extension is *multi-head attention*, where several attention

Mutli-head attention

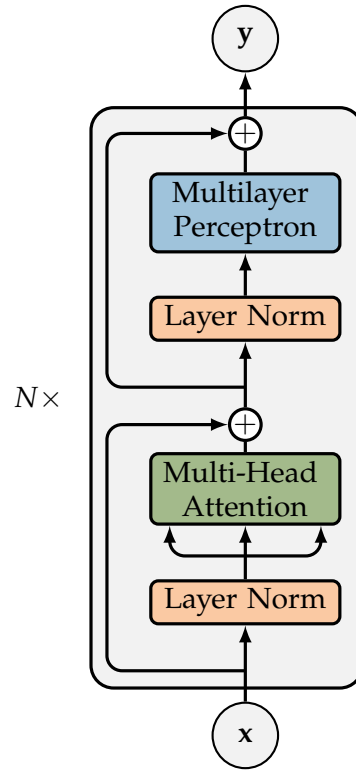


Figure 2.10: Transformer encoder as used in [Dos+20]. The MLP block is applied per element in the token sequence.

operations are performed in parallel with different learned projection matrices. The outputs of each head are concatenated and again multiplied with a learned weight matrix to form the final output of the attention layer.

The *transformer* is a higher-level building block consisting of stacked multi-head self-attention, layer normalization and fully-connected layers [Vas+17]. Originally developed as an encoder-decoder architecture for natural language processing tasks, only the encoder part is used inside ViT. The structure of the transformer encoder can be seen in Figure 2.10. Like standalone self-attention the transformer encoder maps an input sequence of embeddings $x \in \mathbb{R}^{n \times d}$ to an output sequence y of the same length. To extract an embedding sequence from an image, ViT splits the image into flattened patches, linearly transforms them with a learnable matrix and adds positional encodings to each patch such that the transformer can retain information of the relative positions of individual patches. The transformed sequence can then be used for downstream tasks such as classification.

Transformer

Compared to CNNs, attention based models such as ViT can capture long-range relationships inside an image, however they generally require more data to train. There are many improvement and variants of ViT as described above, including hybrid models that employ convolutional and transformer layers [Liu+21; Liu+22; Guo+22].

2.2.3 Loss functions

The *loss function* quantifies the discrepancy between a model's prediction $\mathbf{y}$ and the desired output $\hat{\mathbf{y}}$ and is minimized during training. The concrete choice of loss function not only depends on the task which the network is supposed to solve, but in some cases also influences the training dynamics. In many applications, composite loss functions are formed by combining task-specific loss terms with regularization terms.

2.2.3.1 Regression losses

For image-to-image tasks, e.g., deblurring, denoising, super-resolution, etc., regression losses that compare the network output and a target image per-pixel are common. Let N denote the number of elements (pixels), and let $y_i, \hat{y}_i$ be the i -th elements. The most common regression losses are

- mean squared error ([MSE](#)), also known as L_2 loss

$$\mathcal{L}_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2. \quad (2.63)$$

[MSE](#) is sensitive to outliers and tends to produce smooth solutions.

- mean absolute error ([MAE](#)), also known as L_1 loss

$$\mathcal{L}_{\text{MAE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \|\mathbf{y} - \hat{\mathbf{y}}\|_1. \quad (2.64)$$

[MAE](#) is more robust to outliers than [MSE](#), but has a non-smooth gradient at zero, which can become problematic during optimization.

- Huber loss [[HUB64](#)] is a middle ground between [MSE](#) and [MAE](#). The loss is quadratic near zero and becomes linear in the tails:

$$\mathcal{L}_{\text{Huber}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^N \begin{cases} \frac{1}{2} (y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta, \\ \delta (|y_i - \hat{y}_i| - \frac{1}{2}\delta), & \text{otherwise} \end{cases} \quad (2.65)$$

- Perceptual losses [[JAF16](#)] do not compare the output image to the target image pixel-by-pixel, but instead compare features extracted by another pre-trained, fixed neural network ϕ :

$$\mathcal{L}_{\text{perc}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N_\phi} \|\phi(\mathbf{y}) - \phi(\hat{\mathbf{y}})\|_2^2, \quad (2.66)$$

where N_ϕ is the number of feature elements, which might be different than the number of pixels. This network can be trained for

a completely different task, e. g., [JAF16] use features extracted from the first few layers of a network that was actually trained for image classification.

Perceptual losses are often combined with pixel-wise losses. This generally yields higher perceptual quality to humans than MSE or MAE alone.

2.2.3.2 Classification losses

Classification losses are used if the task of the network is to predict discrete categories or classes. These losses also apply to semantic segmentation of images, in which case they are computed per-pixel.

- Binary cross-entropy is used if there are only two distinct classes, or in multilabel classification, where predicted classes are not mutually exclusive. For a binary label, i. e., $\hat{y}_c = 1$ means the target is of class c and $\hat{y}_c = 0$ is not of class c , it is defined as

$$\mathcal{L}_{\text{BCE}}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{C} \sum_{c=1}^C \hat{y}_c \log(y_c) + (1 - \hat{y}_c) \log(1 - y_c) \quad (2.67)$$

This expects $\mathbf{y}$ to have values between 0 and 1, which is usually achieved with a sigmoid activation after the last layer of the network (see equation (2.38)).

- Cross-entropy is used for multi-class classification with mutually exclusive classes, i. e., $\hat{y}_c = 1$ for exactly one c .

$$\mathcal{L}_{\text{CE}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c=1}^C \hat{y}_c \log(y_c) \quad (2.68)$$

This expects $\mathbf{y}$ to be the output of a softmax layer (see equation (2.41)), where each y_c can be interpreted as the probability that the input of the network belongs to that class.

2.2.3.3 Regularization losses

Regularization terms are independent of the target labels and often encode some desired attributes of the neural network weights $\boldsymbol{\theta}$ themselves. This limited parameter space reduces overfitting and improves generalization. The two most common regularization terms are L2 regularization decay, also known as weight decay $\mathcal{L}_{\text{L2}}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$, which penalizes large weights and L1 regularization $\mathcal{L}_{\text{L1}}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$, which encourages sparsity.

2.2.4 Model training and evaluation

The available data is split into a training set, a test set and optionally a validation set. The data in the *training set* is used for the actual

Dataset splitting

optimization. The *test set* is *only* used to evaluate the generalization capability of a fully trained model and test set data must at no point be used during the training process. A *validation set* is separate from the training set and is sometimes used as a proxy of the test set during training. Elements of the validation set are not directly used inside the optimization algorithms, but periodically checking the performance on the validation set after every epoch is useful for hyperparameter tuning and terminating the optimization algorithms early. A large performance discrepancy computed over the training set, validation set and test set indicates *overfitting*.

Overfitting

Given a training set $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, $i = 1, \dots, n$, the training objective is to find optimal parameters

$$\theta^* = \operatorname{argmin}_{\theta} \frac{1}{n} \sum_i \mathcal{L}(\mathcal{G}(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}). \quad (2.69)$$

This is a very high-dimensional, non-convex optimization problem that is approximated iteratively by gradient-based methods that require the computation of the gradient of the loss function with respect to the network parameters.

2.2.4.1 Gradient-based methods

The simplest gradient-based method is stochastic gradient descent ([SGD](#)), with the iterative update

Stochastic gradient descent

$$\theta_{t+1} = \theta_t - \tau \nabla_{\theta} E_t(\theta_t), \quad (2.70)$$

where the mini-batch objective is

$$E_t(\theta_t) = \frac{1}{|B_t|} \sum_{b \in B_t} \mathcal{L}(\mathcal{G}(\mathbf{x}^{(b)}; \theta_t), \mathbf{y}^{(b)}), \quad (2.71)$$

and θ_0 is initialized with random values, whose distributions should be chosen depending on the chosen activation functions [[He+15](#); [GB10](#)]. In practice, since it is infeasible to compute the gradient of the sum over the whole training set at each iteration, the gradients are instead estimated over random subsets B_t of training samples, called *mini-batches*. The iteration is repeated until convergence criteria are met, e.g., $\|\nabla_{\theta} E_t(\theta_t)\| \approx 0$, or when the validation loss increases between epochs.

The *learning rate* τ is a hyperparameter that determines the convergence speed of the algorithm. In the basic gradient descent method, the learning rate remains fixed over the course of the training but many more advanced variants modify the learning rate with a multiplier at each step based on previous iterations. The next sections will outline the most common [SGD](#) modifications used in deep learning.

Learning rate

MOMENTUM The tendency of [SGD](#) to oscillate inside local minima can be reduced by using momentum [[RHW86](#)], which encourages the weights to move into directions which are similar to previous iterations, like a ball rolling down a slope.

$$\mathbf{u}_{t+1} = \alpha \mathbf{u}_t - \tau \nabla_{\theta} E_t(\boldsymbol{\theta}_t) \quad (2.72)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{u}_{t+1}, \quad (2.73)$$

with $\alpha \in (0, 1)$ controlling the contribution of previous gradients on the update direction. Many other advanced gradient descent methods also use momentum in some form.

ADAGRAD Compared to classic gradient descent, AdaGrad introduces a per-parameter learning rate that is updated over the course of the training [[DHS11](#)]. With $\mathbf{s}_0 = \mathbf{0}$:

$$\mathbf{u}_{t+1} = \nabla_{\theta} E_t(\boldsymbol{\theta}_t) \quad (2.74)$$

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \mathbf{u}_{t+1}^2 \quad (2.75)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \tau \frac{\mathbf{u}_{t+1}}{\sqrt{\mathbf{s}_{t+1} + \epsilon}}, \quad (2.76)$$

where all operations on vectors are element-wise, and $\epsilon > 0$ is a small constant to prevent division by 0. Intuitively, $\mathbf{s}_t$ is a vector that keeps track of how much the corresponding parameter has been updated during training. Dividing the update direction by $\mathbf{s}_t$ decays the global learning rate τ of frequently updated parameters over the course of the optimization more than infrequent ones. AdaGrad works well for sparse gradients but $\mathbf{s}_t$ grows monotonically, which can lead to overly small effective step sizes later in training.

ADAM Adam [[KB15](#)] combines momentum with per-parameter adaptive step sizes and uses moving averages (with a bias correction term) to avoid the monotone step size decay of AdaGrad. With $\mathbf{u}_0 = \mathbf{v}_0 = \mathbf{0}$, the update scheme is

$$\mathbf{g}_{t+1} = \nabla_{\theta} E_t(\boldsymbol{\theta}_t) \quad (2.77)$$

$$\mathbf{u}_{t+1} = \frac{\beta_1 \mathbf{u}_t + (1 - \beta_1) \mathbf{g}_{t+1}}{1 - \beta_1^{t+1}} \quad (2.78)$$

$$\mathbf{v}_{t+1} = \frac{\beta_2 \mathbf{v}_t + (1 - \beta_2) \mathbf{g}_{t+1}^2}{1 - \beta_2^{t+1}} \quad (2.79)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \tau \frac{\mathbf{u}_{t+1}}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \quad (2.80)$$

where again all operations on vectors are element-wise, $\beta_1, \beta_2 \in [0, 1)$ are hyperparameters (commonly chosen $\beta_1 = 0.9$ and $\beta_2 = 0.999$), and $\epsilon > 0$ is a small constant to prevent division by 0. The exponential moving averages $\mathbf{u}_t$ and $\mathbf{v}_t$ estimate the first and second moment of the gradients.

An important variant of Adam is AdamW [LH17], which decouples weight decay from the moment estimation. Instead of adding an L2 regularization loss term as describes as in Section 2.2.3.3, AdamW explicitly adds weight decay to the update step. This changes the last line of the update scheme to

$$\theta_{t+1} = \theta_t - \tau \left(\frac{u_{t+1}}{\sqrt{v_{t+1}} + 1} + \lambda \theta_t \right), \quad (2.81)$$

where $\lambda > 0$ controls the strength of the weight decay.

Adam (especially AdamW with decoupled weight decay) is a robust default and often converges quickly with minimal hyperparameter tuning. However, for many vision tasks using CNNs, SGD with momentum and well-chosen hyperparameters can improve generalization performance compared to adaptive methods [Wil+17].

In addition to the adaptive step sizes inherent to the algorithms above, a *learning rate schedule* that modulates the step size directly, i. e., replacing τ with a step dependent τ_t , is typically used. Common schedule choices include step decay, exponential decay or cosine annealing [LH16].

*Learning rate
schedules*

2.2.4.2 Backpropagation

Backpropagation is an algorithm to efficiently compute the partial derivatives that form the gradients $\nabla_{\theta} \mathcal{L}(\mathcal{G}(x; \theta), y)$ that are required in gradient-based methods by efficiently applying the chain rule. It is a special case of reverse-mode accumulation in the broader field of automatic differentiation [GW08].

The principle of backpropagation is best illustrated on simple feed-forward networks which are just deeply composed functions that form a linear computation graph with the individual layers as nodes

$$\mathcal{L}(\mathcal{G}(x; \theta), y) = \mathcal{L}(f_L(f_{L-1}(\dots f_1(x; \theta_1); \dots); \theta_{L-1}); \theta_L), y). \quad (2.82)$$

Denote the output of the l -th layer as $z_l := f_l(z_{l-1}; \theta_l)$, with $z_l \in \mathbb{R}^{N_l}$ and $\theta_l \in \mathbb{R}^{D_l}$, allowing $D_l = 0$ for layers without learnable parameters, like activation layers. Using the chain rule, the derivative of the loss function with respect to the weights of layer l can be expressed recursively as

Chain rule

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial z_l} \frac{\partial z_l}{\partial \theta_l} \quad (2.83)$$

$$\frac{\partial \mathcal{L}}{\partial z_{l-1}} = \frac{\partial \mathcal{L}}{\partial z_l} \frac{\partial z_l}{\partial z_{l-1}} \quad (2.84)$$

Equation (2.83) is the derivative with respect to the weights of that layer, while equation (2.84) describes how the error propagates backward to the previous layer. The derivatives in these expressions are in general not scalar values but vectors and matrices: Since the output

of $\mathcal{L}$ is scalar, $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_l}$ is a $1 \times N_l$ row vector. $\frac{\partial \mathbf{z}_l}{\partial \theta_l}$ and $\frac{\partial \mathbf{z}_l}{\partial \mathbf{z}_{l-1}}$ are $N_l \times D_l$ and $N_l \times N_{l-1}$ Jacobian matrices, respectively.

The backpropagation algorithm consists of two main phases

1. **Forward Pass:** Compute the output $\mathcal{G}(\mathbf{x}; \theta)$ and the loss $\mathcal{L}$ while storing all intermediate activations $\mathbf{z}_l$.
2. **Backward Pass:** Starting with $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_L}$, recursively compute the gradients $\frac{\partial \mathcal{L}}{\partial \theta_l}$ for each layer using [equation \(2.83\)](#) and propagate the error backward through the network using [equation \(2.84\)](#).

Constructing the full Jacobian matrices is typically unnecessary. For example, the Jacobian matrix of an element-wise activation layer $f_l(\mathbf{z}_{l-1}; \theta_l) = \phi(\mathbf{z}_{l-1})$ would be a $N_{l-1} \times N_{l-1}$ diagonal matrix, with most elements set to zero, and it is more efficient to propagate the derivatives with element-wise multiplication instead of building the Jacobian matrix first. In a fully connected layer $f_l(\mathbf{z}_{l-1}; \theta_l) = \mathbf{W}_{\theta_l} \mathbf{z}_{l-1}$ where the weights θ_l are arranged in a matrix $\mathbf{W}_{\theta_l} \in \mathbb{R}^{N_{l-1} \times N_l}$, the Jacobian $\frac{\partial \mathbf{z}_l}{\partial \theta_l}$ would have $N_{l-1} \cdot N_l^2$ elements and fully instantiating it would quickly exceed memory constraints for large input sizes. Instead, efficient computations are possible via vector-Jacobian products (VJPs), which enable a matrix-free computation of the linear operator the Jacobian represents. The concrete implementation of the VJP depends on the type of layer. For instance, for fully-connected layers the derivatives in [equations \(2.83\)](#) and [\(2.84\)](#) reduce to [\[GWo8\]](#)

*Vector-Jacobian
product*

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{\theta_l}} = \mathbf{z}_{l-1} \frac{\partial \mathcal{L}}{\partial \mathbf{z}_l} \quad (2.85)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_{l-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_l} \mathbf{W}_{\theta_l}^\top. \quad (2.86)$$

Note, that the result of [equation \(2.85\)](#) is now in a different shape, i. e., a $N_{l-1} \times N_l$ matrix instead of a $1 \times D_l$ row vector (where $D_l = N_{l-1} N_l$) as in [equation \(2.83\)](#). However, the individual elements of the result stay the same, they are just arranged differently. By efficiently applying the chain rule without explicitly forming large Jacobian matrices, backpropagation enables efficient gradient computation, and is practical even for deep networks.

Many layers like the ReLU activation function or max-pooling layers are not differentiable everywhere. Deep learning frameworks like PyTorch still assign gradients to points at which the functions are not differentiable, e. g., $\nabla \text{relu}(0) := 0$. In practice, single points at which functions are non-differentiable or even discontinuous usually do not influence the performance of a model.

Backpropagation can become unstable for very deep networks. The repeated multiplication of gradients can result in progressively smaller or larger gradients in earlier layers compared to layers closer to the network's output, e. g., vanishing gradients when using sigmoid activations, because the gradient of the sigmoid function is strictly

*Vanishing and
exploding gradients*

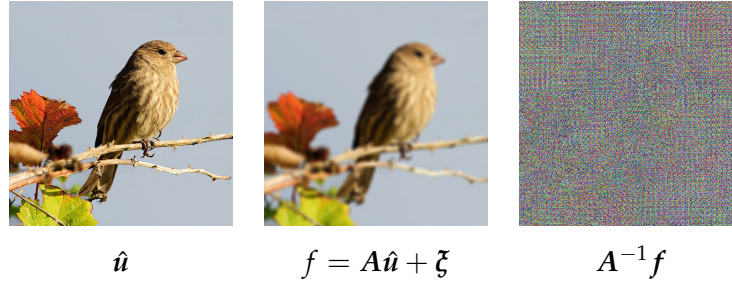


Figure 2.11: Deblurring by direct inversion of the blur operator. Even for very small noise levels, direct inversion amplifies the noise and fails to reconstruct the image.

limited to the interval $(0, 1)$. Techniques like normalization layers (Section 2.2.2.4), using ReLU activations (Section 2.2.2.1), or residual connections (Section 2.2.2.5) can alleviate these issues.

2.3 INVERSE PROBLEMS IN IMAGING

Section 2.1 outlined many error sources that corrupt the measurement of real-world signals during the image formation process, including blur, noise, and lens aberrations. In many image reconstruction tasks, including deblurring, super-resolution, denoising, demosaicing, and MRI- or CT-reconstruction, the observation f can be modeled as the result of the transformation of a clean image $\hat{u}$ under a known (linear) operator A and unknown additive noise ζ :

$$f = A\hat{u} + \zeta \quad (2.87)$$

This section deals with methods to solve the inverse problem of equation (2.87), i. e. given an observed image f , the objective is to reconstruct the clean image $\hat{u}$. The inverse problem is typically ill-posed. A may be non-invertible or poorly conditioned, so small perturbations in the measurement produce large changes in $A^{-1}f$. For instance, consider the task of removing defocus blur in an image. The forward operator A can be approximately modeled as a convolution with a Gaussian kernel whose size depends on the Circle of Confusion (CoC). As depicted in Figure 2.11, even for noise levels that are invisible to the human eye, naively inverting the blur operator fails to reconstruct the true image. This motivates alternative approaches, which will be outlined throughout the rest of this section.

ill-posedness

2.3.1 Variational methods

A classic approach to solve inverse problems is to formulate it as an energy minimization problem derived from the maximum a-priori (MAP) estimate

$$\hat{\mathbf{u}} = \operatorname{argmax}_{\mathbf{u}} p(\mathbf{f} | \mathbf{u})p(\mathbf{u}) \quad (2.88)$$

$$= \operatorname{argmin}_{\mathbf{u}} -\log(p(\mathbf{f} | \mathbf{u})) - \log(p(\mathbf{u})) \quad (2.89)$$

$$:= \operatorname{argmin}_{\mathbf{u}} H_f(\mathbf{A}\mathbf{u}) + \alpha R(\mathbf{u}), \quad (2.90)$$

where the data fidelity term H_f is given by the negative log-likelihood (up to an additive constant), and the regularizer R is derived from the negative log-prior. The weighting parameter α reflects the relative scaling between the two terms. The objective function $E(\mathbf{u}) = H_f(\mathbf{A}\mathbf{u}) + \alpha R(\mathbf{u})$ is sometimes called *energy function*. The *data fidelity term* is determined by the noise model, e. g., assuming independent Gaussian noise

Data fidelity

$$p(\mathbf{f} | \mathbf{u}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\|\mathbf{A}\mathbf{u} - \mathbf{f}\|_2^2}{2\sigma^2}\right) \quad (2.91)$$

results in the common L2 data fidelity term (the factor $\frac{1}{\sigma^2}$ is absorbed in α)

$$H_f(\mathbf{A}\mathbf{u}) = \frac{1}{2}\|\mathbf{A}\mathbf{u} - \mathbf{f}\|_2^2. \quad (2.92)$$

The *regularization function* R encodes prior knowledge about the solution, e. g., the popular total variation (TV) regularization $R(\mathbf{u}) = \|\nabla \mathbf{u}\|_1$ [ROF92] assumes that natural images are approximately piecewise constant.

Regularization function

Proximal algorithms are a popular method to optimize [equation \(2.90\)](#). While there are many variants, whose choice depends on the properties of H_f and R (e. g. differentiability), relevant to this thesis is only the *proximal gradient descent* algorithm, which will be explained in the next sections.

2.3.1.1 Mathematical preliminaries

This section will briefly outline common definitions and theorems useful in convex optimization. For proofs and a more complete overview refer to [RW98; PB+14].

The convergence criteria of convex optimization algorithms require the definition of a few function properties:

Definition 1 (Lipschitz continuity) A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is called *Lipschitz continuous* with constant L if for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ it holds that

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\|_2 \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|_2 \quad (2.93)$$

Lipschitz continuity means that a function can not become arbitrarily steep. If $L < 1$ or $L \leq 1$ then f is also called *contractive* or *non-expansive* respectively.

Definition 2 (L-smoothness) A scalar function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is called *L-smooth*, if it is differentiable and its gradient ∇f is Lipschitz continuous with constant L , i.e.,

$$\|\nabla f(\mathbf{x}_1) - \nabla f(\mathbf{x}_2)\|_2 \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|_2 \quad (2.94)$$

Definition 3 (Convex sets) A subset $\mathcal{C}$ of $\mathbb{R}^n$ is called *convex*, if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}$ and $\alpha \in [0, 1]$

$$(1 - \alpha)\mathbf{x}_1 + \alpha\mathbf{x}_2 \in \mathcal{C} \quad (2.95)$$

Intuitively, for any two points in a convex set $\mathcal{C}$ it also contains the whole line between the two points.

Definition 4 (Convex functions) A scalar function $f : \mathcal{C} \rightarrow \mathbb{R}$ is called *convex* if $\mathcal{C}$ is convex and for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}$ and $\alpha \in [0, 1]$

$$f((1 - \alpha)\mathbf{x}_1 + \alpha\mathbf{x}_2) \leq (1 - \alpha)f(\mathbf{x}_1) + \alpha f(\mathbf{x}_2) \quad (2.96)$$

This definition means that the *epigraph* of a convex function is a convex set. Convex functions are especially useful, because any *local* minimum implies a *global* minimum. The sum of two convex functions is also convex.

Instead of defining a convex function over a convex set, it is sometimes preferred to extend its domain to all of $\mathbb{R}^n$ and set it to $+\infty$ for all inputs outside of $\mathcal{C}$. The *effective domain* of f is the original convex set the function was defined on:

Definition 5 (Effective domain) Let $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$ be a convex function. The *effective domain* of f is defined as

$$\text{dom } f = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) < \infty\} \quad (2.97)$$

Using convex functions that take values in the extended real numbers are useful to encode constraints on the feasible domain inside a regularization function without restricting the input domain, which simplifies some calculations [RW98].

Central to proximal algorithms is the proximal operator of a convex function:

Definition 6 (Proximal operator) Let $f : \mathbb{R} \rightarrow \mathbb{R} \cup \{\infty\}$ be convex and lower semi-continuous. The *proximal operator* of f is defined as

$$\text{prox}_f(\mathbf{x}) = \underset{\mathbf{u}}{\text{argmin}} f(\mathbf{u}) + \frac{1}{2}\|\mathbf{x} - \mathbf{u}\|_2^2 \quad (2.98)$$

Proximal operators are generalized projections. If $\mathbf{x}$ lies outside of $\text{dom } f$, the proximal operator moves it into the feasible region and closer to the minimum. When $\mathbf{x}$ is inside $\text{dom } f$ the proximal operator computes the point that minimizes f while staying close to $\mathbf{x}$. Therefore, for small step sizes, this update acts similarly to a gradient descent step, i. e. $\text{prox}_{\tau f}(\mathbf{x}) \approx \mathbf{x} - \tau \nabla f(\mathbf{x})$ [PB+14]. The proximal operator of some functions have a closed form, e. g., the proximal operator of the L1 norm is the soft-thresholding operator [PB+14]

$$(\text{prox}_{\tau \|\cdot\|_1}(\mathbf{x}))_i = \begin{cases} x_i - \tau & \text{if } x_i \geq \tau \\ 0 & \text{if } |x_i| < \tau \\ x_i + \tau & \text{if } x_i \leq -\tau \end{cases} \quad (2.99)$$

Other, seemingly similar functions have no closed form solution, e. g., the TV regularization $R(\mathbf{u}) = \|\nabla \mathbf{u}\|_1$.

2.3.1.2 Proximal gradient descent

The choice of optimization method to find a minimizer of $E(\mathbf{u}) = H_f(\mathbf{A}\mathbf{u}) + R(\mathbf{u})$ depends on the differentiability of H_f and R , and whether their proximal operator or convex conjugate is easy to compute. If H_f and R are both differentiable, gradient descent as explained in Section 2.2.4 can be used. If E is L-smooth and the step size is chosen with $\tau \in (0, \frac{2}{L})$, gradient descent is guaranteed to converge to a minimizer of E (if it exists) [RB16, page 16]. However, many common regularization functions, like TV regularization, are *not* differentiable or L-smooth. In that case, gradient descent has no convergence guarantees and the iterations oscillate around the minimum.

The proximal gradient descent method, or *forward-backward splitting*, can be used to solve minimization problems of the form $E(\mathbf{u}) = F(\mathbf{u}) + G(\mathbf{u})$, where F and G are convex but only F is L-smooth. It consists of a gradient descent step on F followed by a proximal step on G

Proximal gradient method

$$\mathbf{u}_{t+1} = \text{prox}_{\tau G}(\mathbf{u}_t + \tau \nabla F(\mathbf{u}_t)). \quad (2.100)$$

Like gradient descent, the proximal gradient method converges for a step size $\tau \in (0, \frac{2}{L})$ for any starting point $\mathbf{u}_0$ to a minimizer of E , if such a minimizer exists. For the energy function $E(\mathbf{u}) = H_f(\mathbf{A}\mathbf{u}) + R(\mathbf{u})$ and after applying the chain rule the update becomes

$$\mathbf{u}_{t+1} = \text{prox}_{\tau R}(\mathbf{u}_t + \tau \mathbf{A}^\top \nabla H_f(\mathbf{A}\mathbf{u}_t)). \quad (2.101)$$

The proximal gradient method is most efficient if the proximal operator of R is easy to compute and ideally has a closed-form solution, which is not the case for many popular regularizers, including TV. More complex proximal algorithms that work in more general conditions

exist, like the *alternate direction method of multipliers* (ADMM) [PB+14] or the *primal-dual hybrid gradient* (PDHG) method [CP11], where neither F nor G need to be differentiable.

2.3.2 Deep learning and hybrid methods

Deep neural networks (see Section 2.2) can be employed to directly learn a parameterized function that maps a measured image f directly to a solution $\hat{u} = \mathcal{G}(f; \theta)$ of the inverse problem. Deep learning approaches have gained great success and have been shown to achieve state-of-the-art results on a wide array of inverse problems in imaging. However, compared to classical methods, learning based methods require a large amount of training data, i. e., inputs f_i for which the desired output $\hat{u}_i$ is already known. The training process itself is resource intensive, which makes it costly to adapt the network to a different task.

To combine the advantages of powerful feed-forward networks and model-based approaches, authors have considered various hybrid models like learning regularizers (e. g. [RB09; AEB06; HKD13; CRP14]), designing network architectures that resemble the structure of minimization algorithms or differential equations, e. g., [SR14; Zhe+15; Kob+17; CYP15], interleaving networks with classical optimization steps [Dia+21; Liu+18a; Liu+18b], or using the parameterization of networks as a regularization for equation (5.2) [UVL18; Hec+19].

Particularly relevant hybrid methods in this thesis are *plug-and-play priors*. These approaches are based on the fact that any Gaussian denoising algorithm represents a proximal operator [Hei+14; VBW13; DKE10]. Gaussian denoising itself can be stated as an inverse problem where A is the identity. Then, combining the definition of Gaussian noise (equation (2.91)) with the MAP estimate (equation (2.89)) yields

Plug-and-play priors

$$\hat{u} = \operatorname{argmax}_u p(f | u) p(u) \quad (2.102)$$

$$= \operatorname{argmin}_u -\log(p(f | u)) - \log(p(u)) \quad (2.103)$$

$$= \operatorname{argmin}_u \frac{1}{2\sigma^2} \|u - f\|_2^2 - \log(p(u)) \quad (2.104)$$

$$= \operatorname{prox}_{-\frac{1}{\sigma^2} \log(p(\cdot))}(f). \quad (2.105)$$

This means that the Gaussian denoiser encodes the proximal operator of some prior and can be used for regularization [Hei+14], giving rise to plug-and-play algorithmic scheme

$$u_{t+1} = \mathcal{D} \left(u_t + \tau A^\top \nabla H_f(Au_t) \right), \quad (2.106)$$

where $\mathcal{D}$ denotes the denoiser. This denoiser can even be a neural network $\mathcal{D} \equiv \mathcal{G}(\cdot; \theta)$ that only needs to be trained once for Gaussian denoising and can be reused for multiple inverse problems by simply changing the forward operator in equation (2.106), combining

advantages from both deep learning and energy minimization while achieving superior performance compared to handcrafted [Mei+17]. However, a drawback of this approach is that it sacrifices the provable convergence property of the classical proximal gradient method. The algorithmic scheme in equation (2.106) is only guaranteed to converge for any starting position if the neural network is non-expansive, i. e., if it is 1-Lipschitz continuous [Ryu+19]. Although there are Lipschitz normalization techniques for neural networks [Gou+21; SGL18], they are very restrictive for deep networks and limit the denoising performance. A solution for this will be developed in Chapter 5.

2.4 POINT-BASED RENDERING

Computer graphics aims to generate images of virtual or captured 3D scenes by simulating the propagation of light and its interaction with surfaces. The process of computing a 2D image from a 3D description of a scene is referred to as *rendering*. In most rendering applications, surfaces are represented by polygonal meshes, typically consisting of triangles, which are then rasterized or ray-traced to produce the rendered image.

Computer graphics

In contrast, in *point-based rendering*, surfaces are represented as a collection of discrete point samples, which are not explicitly interconnected. In addition to their position, points can carry additional geometric and appearance attributes, which are used by the renderer to construct an image. This representation is particularly attractive for data acquired from 3D sensors, such as ToF cameras (see Section 2.1.2.6) or LiDAR, where measurements are naturally given as point clouds rather than meshes.

Point-based rendering

The lack of topology information makes point clouds very lightweight and enables applications such as 3D reconstruction and SLAM algorithms [Kel+13; Whe+15; LWK15], mobile robotics [PCS+15], tele-immersion [MBC16; DB07] and the visualization of large scale terrestrial data [Don+20; Bie+07]. However, the missing explicit storage of connectivity information can also be a drawback and make tasks such as surface reconstruction, normal estimation, and rendering harder, since local surface information must first be inferred from the point distribution.

2.4.1 Point clouds

In this dissertation, a *point cloud* is defined as a finite set of 3D samples

$$\mathcal{P} = \{(x_i, \mathbf{n}_i, r_i, \mathbf{a}_i)\}_{i=1}^N, \quad (2.107)$$

where $x_i \in \mathbb{R}^3$ denotes the position of the i -th point, $n_i \in \mathbb{R}^3$ is the point's surface normal, $r_i > 0$ is a local *radius* that represents the spatial support of the point (e. g., an estimate of the local sampling density or surfel size), and $a_i \in \mathbb{R}^A$ is a collection of additional associated attributes such as color, reflectance, or confidence values.

Unlike triangle meshes, point clouds contain no explicit topology information. Although a simple rendering algorithm could project each point to a single pixel, depending on the density of the point cloud, this would result in sparse images with many empty areas. Therefore, rendering algorithms must reconstruct an (at least visually convincing) continuous surface from these unstructured samples. While there are methods to convert point clouds to triangle meshes, these are usually costly, thus the next sections will explain surfel-based techniques that are able to render point clouds directly.

2.4.2 Rendering opaque splats

A straightforward way to visualize a point cloud is to project each point onto the image plane and draw a small opaque *splat* at the projected position. In the simplest case, this splat is a small disk or a square aligned according to the surface normal n_i and whose size is derived from the radius r_i . Each splat is projected to a set of pixels. A naive renderer assigns a color or intensity value to all pixels covered by the splat's projected footprint based on its normal n_i , attributes a_i and the shading model (e. g. phong). If multiple points contribute to the same pixel, i. e., if splats overlap, a depth test is employed to keep only the contribution of the splat closest to the camera, effectively treating splats as opaque surface elements.

Splats

The rendering of opaque splats can be trivially implemented in modern graphics APIs (e. g., OpenGL, Vulkan), by just rendering a disk at each point. Due to their speed and simplicity, opaque splats are used in several point-based 3D reconstruction methods to render the current model into color and depth images for camera pose estimation [Kel+13; Whe+15; LWK15]. Still, this approach has several drawbacks:

- Large opaque splats tend to over-smooth fine geometric detail and cause *halo* artifacts at depth discontinuities.
- The perceived surface smoothness and completeness depend on the chosen radii r_i , but even for small splats the hard splat boundaries generally result in a non-smooth appearance.
- The method is sensitive to outliers and noisy point positions or normals, which can appear as isolated splats and other artifacts.

For these reasons, opaque splats are not well suited for the visualization of very noisy point clouds, like unprocessed point clouds produced with ToF cameras, which are prone to flying pixels.

2.4.3 EWA splatting

In Elliptical Weighted Average (EWA) splatting [Zwi+01], each point is again treated as a small oriented disk with a finite extent, however, in contrast to opaque splatting, the influence of a splat decreases smoothly towards its edges. Concretely, each position x on a splat is given a weight by a radial basis function $w_i(d) \geq 0$, where $d = \|x - x_i\|$ is the distance to the splat center. For a smooth surface, the scale of the weighting function should be chosen such that it vanishes at the edge of the splat, i. e., $w_i(r_i) \approx 0$. Let $(u, v) \in \mathbb{R}^2$ denote a screen space pixel and $q_i(u, v) \in \mathbb{R}^3$ the intersection of the viewing ray through (u, v) and the plane containing the i -th splat. EWA now computes a weighted average of all splat attributes as

$$a(u, v) = \frac{\sum_{i \in \mathcal{I}_\epsilon} w_i(\|q_i(u, v) - x_i\|) a_i}{\sum_{i \in \mathcal{I}_\epsilon} w_i(\|q_i(u, v) - x_i\|)}. \quad (2.108)$$

Since the weighting function $w_i(d)$ falls to 0 at splat edges, the only splats contributing to the sums in equation (2.108) are the ones intersected by the viewing ray. Additionally, to avoid occluded background surfaces to be included in the computation, a *surface thickness* parameter ϵ is used to discard splats whose intersection with the viewing ray is farther away than ϵ from the intersection closest to the viewer. Equation (2.108) can not only be used to interpolate the splat appearance vector, but also their normals. This enables a per-fragment light computation similar to Phong shading [BSK04].

Surface thickness

Compared to opaque splatting, a hardware-accelerated implementation of EWA splatting in a modern, rasterization-based graphics pipeline is more complicated. Commonly, a 3-pass deferred shading approach consisting of a *visibility pass*, an *accumulation pass* and a *normalization pass* is utilized [Bot+05]. In the *visibility pass*, all splats are moved backward by ϵ along the viewing rays before they are rendered into the depth buffer. The splats are treated as if they were opaque during this pass and no shading is needed.

Hardware-acceleration

During the *accumulation pass*, the scene is rendered with splats in their original position. The weighted attributes $w_i(\|q_i(u, v) - x_i\|) a_i$ and weights $w_i(\|q_i(u, v) - x_i\|)$ are computed in fragment shaders and accumulated in two separate buffers using additive blending. Deactivating writing to the depth buffer (but not the depth test) during this pass in combination with the back-shifted depth buffer from the visibility pass ensures that only contributions up to a distance from the closest splat according to the surface thickness parameter are accumulated. This means, that after the accumulation pass, the two buffers contain the numerator and denominator of equation (2.108). The *normalization pass* computes the attributes $a(u, v)$ with a simple pixel-wise division of the accumulated weighted attributes $\sum_{i \in \mathcal{I}_\epsilon} w_i(\|q_i(u, v) - x_i\|) a_i$ and summed weights $\sum_{i \in \mathcal{I}_\epsilon} w_i(\|q_i(u, v) - x_i\|)$ which are stored in the two

buffers. As this pass only requires per-pixel operations, it does not require a full re-rendering of the scene. The final lighting computations and shading can also be included in the normalization pass.

The original formulations of EWA splatting [Zwi+01; Zwi+02] also include a screen-space pre-filter (usually Gaussian) to avoid aliasing artifacts due to minification, which has been omitted in the explanations above. Approximating the pre-filtering by simply scaling the splats to cover at least 2×2 pixels has been shown to achieve high quality while still being efficient [Bot+05].

Compared with opaque splatting, EWA splatting has an inherent *denoising effect*. By averaging contributions of nearby splats along a viewing ray, small-scale noise in the splat positions is smoothed out. This effect becomes stronger for higher surface thickness values ϵ , but at the cost of increased blurring and mixing of unrelated surfaces. Large outliers can still bias the resulting rendered surface towards the camera. Additionally, there are still halos if a splat overlaps a depth discontinuity. Efficient and outlier-robust solutions to these problems will be developed in Chapter 7.

Problems

3

LEARNING SENSOR LAYOUTS FOR DEEP LEARNING TASKS

Conventionally, image data processing and deep learning applications implicitly assume a fixed, predetermined sensor configuration that is not subject to the training process. More recent work in the literature changes this notion and treats different parameters of the imaging device as part of the deep-learning-based processing pipeline. These parameters are then optimized together with the neural network parameters in an end-to-end fashion in order to tailor them for specific applications. This chapter takes the idea of end-to-end optimization of imaging sensor parameters a step further and presents a novel method to optimize the physical layout and density of pixels in the sensor plane, thus effectively maximizing the *spatial* information value in the most important areas for a task. It derives an analytic, differentiable approach for the sensor layout parameterization, that allows for task-specific, varying pixel resolutions. Two pixel layout parameterization functions are presented: rectangular and curvilinear grid shapes that retain a regular topology. A drop-in module approximates the sensor simulation given existing high-resolution images and directly connects the proposed method with existing deep learning models. Experiments investigating two different downstream tasks, classification and semantic segmentation, show that network predictions benefit from learnable pixel layouts. The approach developed in this chapter was published in [Som+24].

3.1 INTRODUCTION

Deep learning models have achieved impressive performance in a wide range of computer vision tasks, including image classification, object detection, and semantic segmentation. Computational imaging approaches go beyond the traditional paradigm in computer vision, which uses an image as input, and jointly consider the imaging processing method and the image formation process performed in a camera, comprising various hardware design and low-level, on-chip image processing parameters. Recent approaches in deep computational imaging aim at task-specific optimization of ISP pipeline parameters [Mos+20], various optical components [Cha+18; CW19; Chu+21; Met+20; Sun+20; Tse+21], or color filters, as explained in Chapter 4, in an end-to-end manner.

Although the trend toward high-resolution imagery remains unbroken, specifically mobile systems that incorporate computer vision

solutions are strongly focused on cost, energy, and resource efficiency, as well as on data security. Therefore, another trend in deep computational imaging addresses new hardware concepts that integrate edge AI and neural sensor concepts [Itu+23; Suo+23; MW21; Kli+22; NMW22]. These approaches also solve vision problems in an end-to-end, task-specific manner, jointly optimizing the vision algorithm and the imaging hardware parameters [Kli+22]. Here, the primary goal is to sense data with high information value useful for the final task [Suo+23; Itu+23], thus minimizing energy and data communication requirements, as well as the amount of captured data to maximize privacy. Although the integration of AI compute and sensing capabilities is a medium to long-term trend in hardware development [Itu+23], it results in a reduced amount of visual data acquired without losing performance on the task. An example is in-pixel processing with a SCAMP-5 chip with 256^2 px [MW21].

This chapter is motivated by the hardware trend towards integrated AI compute and sensing capabilities [Itu+23], but instead investigates the option to use currently available parameters in standard CMOS sensor design processes. It specifically focuses on non-uniform and minimalist pixel layouts, i. e., the number, location, size, and, potentially, shape of pixels on the image sensor. Hardware designs of such non-uniform pixel layouts were recently proposed [Cho+24; Zha+25]. This objective is in line with the common practice of artificially decreasing the image resolution at the input for network training to adhere to memory constraints on the GPU. To make this flexibility in pixel layout design accessible to deep learning, a data-driven approach is proposed that optimizes the pixel layout for a given task together with the neural network parameters in an end-to-end fashion.

The chapter demonstrates that various tasks, such as autonomous driving, benefit from specifically optimized pixel layouts, especially if the task includes a *spatial bias*, where certain image regions require a higher pixel density than others. The effectiveness of the proposed method is validated for classification and semantic segmentation tasks. A significant improvement over baselines is shown without significant dependence on the specific network architecture utilized to solve the task. The contributions are

- a differentiable, physically based sensor simulation framework that allows end-to-end gradient-based optimization of pixel layouts,
- a generic pixel layout parameterization that covers a large class of possible geometries, including rectangular and free-form pixel shapes, while retaining the regular topology, usually required by downstream networks,

Motivation

Contributions

- a drop-in module, that can approximate the sensor simulation given existing high-resolution images, and can thus be easily incorporated into existing deep learning models, and
- an experimental validation, that shows in simulations that tasks like semantic segmentation in autonomous driving can benefit from non-uniform pixel layouts.

3.2 RELATED WORK

This section discusses prior work that motivates the need for non-uniform pixel layouts, attempts at end-to-end optimization of ISP pipelines for various downstream tasks, as well as superpixel-based methods that end-to-end optimize superpixel generation and network parameters.

3.2.1 Pixel layout

In computer vision tasks like semantic segmentation, most methods employ an “hour-glass model” like UNet [RFB15], PSPNet [Zha+17b], etc. to be computationally efficient. Here, the information is first encoded as low-resolution feature maps and then upsampled to target image resolution for the downstream task at hand. However, this approach makes the strong assumption that all regions in the image have equal information and thus equal importance. In practice, this assumption does not hold, as Jin et al. [Jin+22] show that a network’s performance can benefit from non-regular, learnable downsampling strategies. Additionally, some early works argue for non-regular pixel layouts due to their ability to improve super-resolution [BLW07] or to provide better image representations as such [Kir10].

Recently, Choubey et al. [Cho+24] propose a CMOS sensor design methodology that allows for a non-uniform layout of rectangular pixels across the sensor plane. The method presented in this chapter can find such an optimal layout for a given task and data distribution combination automatically.

3.2.2 End-to-end optimization of the ISP pipeline

Some attempts have been made to jointly optimize the ISP pipeline and network parameters [Mos+20; Jin+22; Mar+19; TM21]. In their work, Mosleh et al. [Mos+20] proposed a “hardware-in-the-loop” method to jointly optimize the hardware ISP and network parameters stated as a multi-objective problem using a 0th-order stochastic solver for perceptual image quality. Talebi and Milanfar [TM21] use a “CNN-based resizer” to downsample high-resolution images and then perform clas-

sification using deep learning-based recognition models and jointly learn weights for both the “resizer” and the recognition model, which is extremely complex, is not buildable as a sensor, and only performs uniform downsampling. Marin et al. [Mar+19] proposed an edge-based downsampling scheme that maps images to a non-uniform pixel layout with a focus on object boundaries, i. e., edges. They learn to sample with a higher resolution near object boundaries and lower resolution elsewhere. This helps in improving the computational cost while retaining some essential information for the task. Nonetheless, many tasks also require information from the object’s interior, e. g., cancer cell detection. To overcome this drawback, Jin et al. [Jin+22] proposed to learn a deformed sampling density distribution used for downsampling a high-resolution image over a non-uniform grid based on the network’s performance on the downstream task rather than the edge vicinity. However, [Jin+22; Mar+19] use a neural network *on the high-resolution image* to predict a suitable pixel layout for each image individually, which requires the neural network to operate on high-resolution data. In contrast, the method presented in this chapter learns the hardware sensor layout with the subsequent network operating on the produced low-resolution data only, considering the full pixel radiance, not just a single sampling position. High-resolution data is only required for training, and there is no computational overhead during inference. The proposed drop-in module can take existing high-resolution images to approximate the incident radiance, allowing to optimize any network for a limited resolution budget, or to reduce the network’s size and inference time while keeping performance as high as possible. In prior work [Ria+22], similar approaches have been taken by *uniformly* decreasing the pixel resolution of the input images to reduce the network size until the performance falls below a given lower bound. The approach in this chapter can thus be seen as a generalization of [Ria+22], integrating non-regular sampling schemes like [Jin+22], while considering the total intensity of the input image (as an approximation of the incident radiance to a sensor).

3.2.3 Superpixel-based methods

In addition to learning sensor parameters for effective downsampling, other approaches learn pixel grouping into larger superpixels together with neural network parameters [YF21; Fey+18; Ave+20]. Yet, all such approaches create *image specific* superpixels and are therefore not suitable for learning a *fixed* hardware sensor layout. Moreover, as inspired by [Cha+18; CW19; Chu+21; Met+20; Mos+20; Sun+20; Tse+21] the objective in this chapter is to optimize sensor parameters (or image downsampling) in a photometrically correct way, i. e., considering the continuous acquisition of radiance at the pixel level as described in [Section 2.1.2](#). Another related pixel is hardware pixel binning [Zha+18;

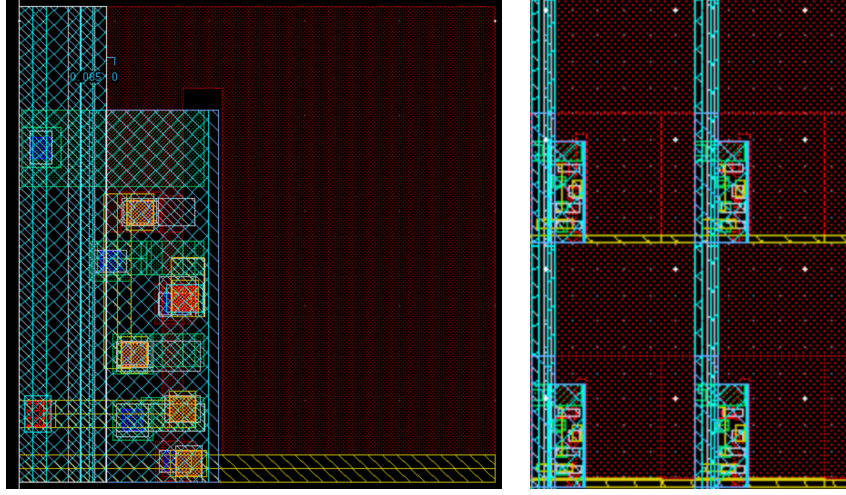


Figure 3.1: Base pixel (left) and non-uniform (right) IC layouts for a section of the designed CMOS image sensor in [Cho+24]. The four pixels in the non-uniform section are vertically expanded. The base pixels are visible in the bottom left corners of each of the four pixels.

YIP15; Wes+09; Cho+14; Men+22], e. g., to reduce noise in low-light resolutions. Typically, these approaches apply uniform binning. Although pixel binning can be used to implement the non-uniform layouts presented later in this chapter in field programmable gate arrays (FPGAs) and allow modifications of the pixel layout without re-manufacturing the sensor, the hardware resource and energy requirements remain unchanged.

3.3 DIFFERENTIABLE SENSOR LAYOUT OPTIMIZATION

In prior work, Choubey et al. [Cho+24] proposed a non-uniform hardware design that includes *rectangular* pixels of different sizes across the sensor plane. The design consists of a square *base pixel* that represents the minimum pixel size, and extends the base pixel horizontally or vertically to enable non-uniformity across the sensor plane (see Figure 3.1). The design space for such layouts is very large and the black-box nature of neural networks make it hard to manually decide on the spatial pixel density across the sensor plane to improve the overall performance. Thus, the goal is to *automatically* find the optimal layout for a given task and network combination.

To this end, unlike usual deep learning techniques, images that are used as input to a task-specific network $\mathcal{G}$ are not considered fixed, but instead functions $I(L_e; \theta)$. The inputs to these functions are a continuous radiance function L_e which depends on the scene to

End-to-end sensor optimization

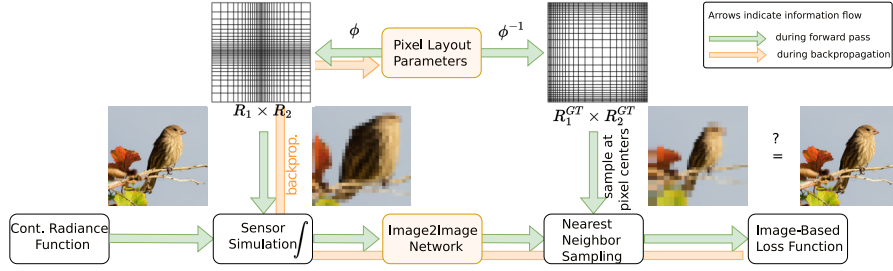


Figure 3.2: The overall pipeline for the end-to-end learning of task-specific pixel layouts. $R_1 \times R_2$ and $R_1^{GT} \times R_2^{GT}$ denote the resolutions of the sensor and reference image, respectively. To best visualize the sizes of individual pixels compared with the input radiance function after resampling, the network in this visualization is just the identity function.

be imaged, as well as layout parameters θ which describe the pixel layout. This naturally requires modeling a spatially continuous and differentiable sensor simulation process that determines the value of each pixel for a given layout described by θ . This allows a gradient based optimization of θ *jointly* with network parameters ν , i. e.,

$$\min_{\theta, \nu} \mathbb{E}_{(L_e, \mathbf{y})} (\mathcal{L}(\mathcal{G}(I(L_e; \theta); \nu), \mathbf{y})), \quad (3.1)$$

where $\mathcal{L}$ is a suitable loss function to compare the prediction of the network with the desired ground truth $\mathbf{y}$.

The overall approach to represent I as a function $I(L_e; \theta)$ is depicted in Figure 3.2 and the individual components are explained in detail in the following sections. Assuming a *continuous radiance function* (which in this work is approximated by a high-resolution image) as input, a *sensor simulation* is performed (see Section 3.3.1) to capture the radiance hitting the individual (possibly deformed) pixels defined by *pixel layout parameters* (see Section 3.3.2). The pixel layout is a continuous, bijective function ϕ applied to a regular pixel grid that retains the pixel topology. Thus, the resulting, usually distorted pixel layout can be fed to any network that accepts images as input. In case of a downstream image-to-image network that uses an image-based loss function, a *back-warping*, i. e., an interpolation using ϕ^{-1} , is applied to resample the network's output to the original image resolution. As the pixel layout model is differentiable, the optimal task-dependent layout can be learned jointly with the network (see Section 3.3.3).

3.3.1 Measurement equation

The basis for the sensor simulation is the general measurement equation derived in Section 2.1.2. The intensity measurement of a single

pixel with multi-index $k = (k_1, k_2)$ is given by [equation \(2.25\)](#). Given separable $W_k(\mathbf{p}, \boldsymbol{\omega}, t, \lambda) = W_{p,k}(\mathbf{p})\tilde{W}_k(\boldsymbol{\omega}, t, \lambda)$ yields

$$\begin{aligned} I_k &= \int_{\lambda_{\min}}^{\lambda_{\max}} \int_{t_{\min}}^{t_{\max}} \int_{A_k} \int_{\Omega} W_k(\mathbf{p}, \boldsymbol{\omega}, t, \lambda) L_{e,\lambda}(\mathbf{p}, \boldsymbol{\omega}, t, \lambda) d\boldsymbol{\omega} d\mathbf{p} dt d\lambda \\ &= \int_{A_k} W_{p,k}(\mathbf{p}) H(\mathbf{p}) d\mathbf{p} \end{aligned} \quad (3.2)$$

where $W_{p,k}$ is a weighting term that models a possible varying responsivity of the sensor with respect to the location inside a pixel and $H(\mathbf{p})$ refers to the weighted radiance integrated over exposure time, hemisphere and wavelengths¹, i. e.

$$H(\mathbf{p}) = \int_{\lambda_{\min}}^{\lambda_{\max}} \int_{t_{\min}}^{t_{\max}} \int_{\Omega} \tilde{W}_k(\boldsymbol{\omega}, t, \lambda) L_{e,\lambda}(\mathbf{p}, \boldsymbol{\omega}, t, \lambda) d\boldsymbol{\omega} dt d\lambda. \quad (3.3)$$

The measurements are then further processed in an [ISP](#) pipeline to compute an RGB, but for simplicity, all the following derivations treat $H(\mathbf{p})$ to directly output RGB values. The final image color is then just the weighted mean over the pixel area.

$$I_k = \frac{1}{\text{area}(A_k)} \int_{A_k} W_{p,k}(\mathbf{p}) H(\mathbf{p}) d\mathbf{p}, \quad (3.4)$$

where the normalization by pixel area is required to allow different pixel sizes without influencing the color and brightness in the final image.

3.3.2 Parameterizing pixel layouts

Although the design in [\[Cho+24\]](#) is limited to rectangular pixels, this section will introduce a more general framework to parameterize pixel layouts, as further advances in sensor design methodologies are expected in the future.

The full sensor area is defined to be the square region $\mathcal{S} = [-1, 1]^2$. As such, each pixel has its own domain $A_k \subset \mathcal{S}$ and weighting functions $W_{p,k}$. Pixels are assumed to be disjoint, except for their boundaries ∂A_k , and their union covers the whole sensor area, i. e.,

$$\mathcal{S} = \bigcup A_k. \quad (3.5)$$

Pixels from the standard, uniform layout are denoted as U_k . In this case the pixel boundaries match the parallel grid lines of the sensor, i. e., for the sensor resolution $R_1 \times R_2$,

$$U_k = \left[\frac{2k_1 - R_1}{R_1}, \frac{2(k_1 + 1) - R_1}{R_1} \right] \times \left[\frac{2k_2 - R_2}{R_2}, \frac{2(k_2 + 1) - R_2}{R_2} \right]. \quad (3.6)$$

¹ Unweighted spectral radiance integrated over time, directions and wavelengths known as *radiant exposure* $H_e = \frac{\partial Q_e}{\partial A}$, which is why H was chosen as the symbol here.

A class of pixel layouts is now defined as a parameterized deformation function $\phi : \mathcal{S} \times \mathcal{D} \rightarrow \mathcal{S}$, where $\mathcal{D} \subset \mathbb{R}^d$ is the set of possible parameters. The function $\phi(\cdot; \theta)$ is required to be bijective and bi-Lipschitz, which implies that ϕ and ϕ^{-1} are differentiable almost everywhere for fixed $\theta \in \mathcal{D}$. For every θ , the function $\phi(\cdot; \theta)$ maps the sensor to itself. The pixels with index k of this layout are defined as the set

$$A_k(\theta) = \phi(U_k; \theta), \quad (3.7)$$

i.e., the image of the corresponding uniform pixel area of index k . The bijectivity constraint of ϕ assures that the deformed pixels do not overlap and the overall pixel number and topology is retained. Since ϕ is Lipschitz, neighboring pixels of the uniform layout are mapped to neighboring pixels in the deformed layout, and boundaries get mapped to boundaries, i.e., $\partial A_k(\theta) = \phi(\partial U_k; \theta)$.

With these definitions, the forward pass of the image formation process, i.e., the pixel color as given by [equation \(3.4\)](#), can be computed by a change of variables as integrals over U_k instead of $A_k(\theta)$

$$\begin{aligned} I_k(\theta) &= \frac{1}{\text{area}(A_k(\theta))} \int_{A_k(\theta)} W_{p,k}(\mathbf{p}; \theta) H(\mathbf{p}) d\mathbf{p} \\ &= \frac{\int_{U_k} W_{p,k}(\phi(\mathbf{u}; \theta); \theta) H(\phi(\mathbf{u}; \theta)) |\det D_u \phi(\mathbf{u}; \theta)| d\mathbf{u}}{\int_{U_k} |\det D_u \phi(\mathbf{u}; \theta)| d\mathbf{u}}. \end{aligned} \quad (3.8)$$

Here, $D_u \phi$ is the 2×2 Jacobian matrix of ϕ with respect to the position argument $\mathbf{u}$. Since in the most general case the weighting function may depend on the pixel layout, θ is explicitly added to its arguments.

There are many possible concrete choices for ϕ , with varying degrees of freedom with respect to θ . Currently, rectangular pixels have an advantage in manufacturing the optimized layout in hardware [[Cho+24](#)]. For this reason, a simple deformation that depends only on two parameters is proposed. Still, experiments show that even such a simple pixel layout already leads to significant improvements. Given $\mathbf{p} = (p_1, p_2) \in \mathcal{S}$ and $\theta = (\theta_1, \theta_2) \in (-1, 1)^2 =: \mathcal{D}$ the layout is defined as

$$\phi_{\text{rect}}(\mathbf{p}; \theta) = \left(\frac{\frac{p_1(\theta_1 - 1)}{2\theta_1|p_1| - \theta_1 - 1}}{\frac{p_2(\theta_2 - 1)}{2\theta_2|p_2| - \theta_2 - 1}} \right). \quad (3.9)$$

The parameters control the vertical and horizontal deformation strengths independently and, based on their sign, move more pixels to the edges or the center of the sensor.

Currently, there are no comprehensive hardware design constraints for pixel layouts, but designs beyond rectilinear ones are possible. Therefore, it is shown, as an example, that the proposed method can

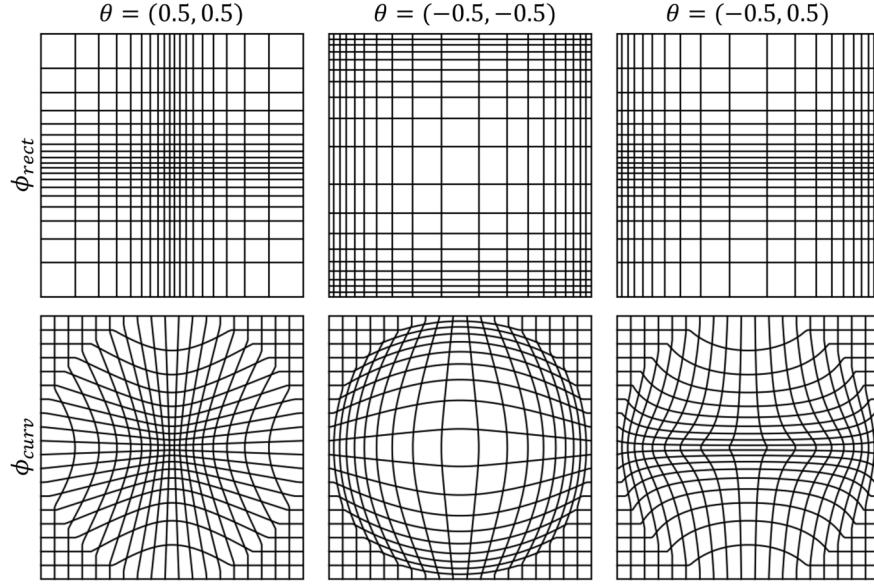


Figure 3.3: The rectangular (top row) and curvilinear (bottom row) pixel layouts used in this chapter and their behavior under different parameters on a 20×20 sensor.

also handle curvilinear layouts with a circular version of [equation \(3.9\)](#), specifically

$$\phi_{\text{curv}}(\mathbf{p}; \boldsymbol{\theta}) = \begin{cases} \begin{pmatrix} \frac{p_1(\theta_1-1)}{2\theta_1\|\mathbf{p}\|_2-\theta_1-1} \\ \frac{p_2(\theta_2-1)}{2\theta_2\|\mathbf{p}\|_2-\theta_2-1} \end{pmatrix} & \text{if } \|\mathbf{p}\|_2 < 1 \\ \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} & \text{otherwise} \end{cases}. \quad (3.10)$$

Hardware designs for these curvilinear layouts exist [[Zha+25](#)]. Visualizations of the rectangular and curvilinear layouts can be found in [Figure 3.3](#).

3.3.3 End-to-end optimization

Since the goal is to optimize the pixel layout for some downstream task, this section will explain how the pixel layout parameterization can be incorporated in a gradient-based optimization scheme. The outputs of the sensor simulation $I_k(\boldsymbol{\theta})$ (see [equation \(3.8\)](#)) are processed by a neural network $\mathcal{G}$, and subsequently input into a task-specific loss function $\mathcal{L}$. To enable gradient based optimization, the gradient of $\mathcal{L}$ with respect to the sensor parameter $\boldsymbol{\theta}$ needs to be computed. Applying the chain rule gives

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \sum_k \frac{\partial \mathcal{L}}{\partial I_k} \frac{\partial I_k}{\partial \theta_j}, \quad (3.11)$$

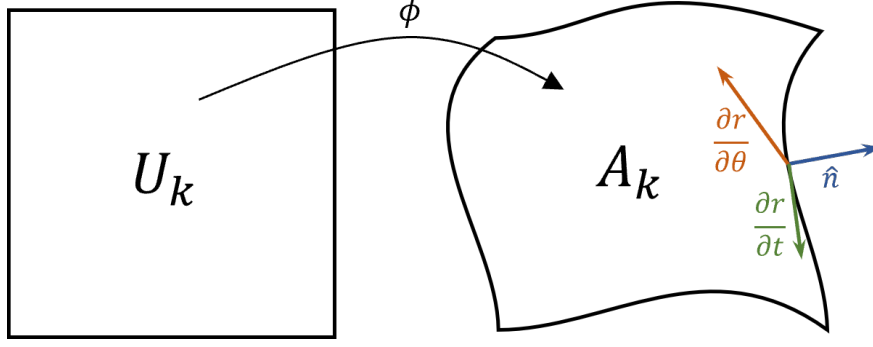


Figure 3.4: Applying ϕ to a uniform pixel U_k yields the deformed pixel A_k , whose shape depends on the parameter vector θ . The flux integral for the gradient computation requires quantities on each boundary point $r \in \partial A_k$: The tangent $\frac{\partial r}{\partial t}$, the outward-pointing normal $\hat{n}$, and the change of the boundary point with respect to the pixel layout parameterization $\frac{\partial r}{\partial \theta}$.

where $\frac{\partial \mathcal{L}}{\partial I_k}$ can be computed via standard backpropagation. In the following, an analytic expression for the second term, i. e. the derivative of a pixel color with respect to the sensor layout parameters is derived. For brevity, the j subscript is dropped, so that θ always refers to a scalar value. However, the calculations can be trivially vectorized.

Starting from [equation \(3.4\)](#), applying the quotient rule yields

$$\frac{\partial I_k}{\partial \theta} = \frac{\partial}{\partial \theta} \left(\frac{1}{\text{area}(A(\theta))} \int_{A_k(\theta)} W_{p,k}(\mathbf{p}; \theta) H(\mathbf{p}) d\mathbf{p} \right) \quad (3.12)$$

$$=: \frac{\partial f(\theta)}{\partial \theta g(\theta)} \quad (3.13)$$

$$= \frac{\frac{\partial f}{\partial \theta}(\theta) g(\theta) - f(\theta) \frac{\partial g}{\partial \theta}(\theta)}{g(\theta)^2}. \quad (3.14)$$

Gradient
computation

The derivative of f will now be explained in detail, i. e.,

$$\frac{\partial f}{\partial \theta} = \frac{\partial}{\partial \theta} \int_{A_k(\theta)} W_{p,k}(\mathbf{p}; \theta) H(\mathbf{p}) d\mathbf{p}. \quad (3.15)$$

The derivative of the pixel area is computed analogously. Since the integration domain itself depends on θ , Reynold's transport theorem [Fla73]² can be applied:

$$\begin{aligned} \frac{\partial f}{\partial \theta} &= \int_{A_k(\theta)} \frac{\partial}{\partial \theta} W_{p,k}(\mathbf{p}; \theta) H(\mathbf{p}) d\mathbf{p} \\ &\quad + \oint_{\partial A_k(\theta)} W_{p,k}(\mathbf{r}; \theta) H(\mathbf{r}) \left\langle \frac{\partial \mathbf{r}}{\partial \theta}, \hat{\mathbf{n}}(\mathbf{r}; \theta) \right\rangle ds \\ &=: Q_{\text{int}}(\theta) + Q_{\text{bound}}(\theta) \end{aligned} \quad (3.16)$$

The first integral can be easily computed by a change of variables like [equation \(3.8\)](#). The second integral is the flux integral across ∂A_k ,

² Also known as the Leibniz integral rule

where $\mathbf{r} : [0, 1] \rightarrow \partial A_k(\theta)$ is an arbitrary piece-wise smooth parameterization of the boundary, $\frac{\partial \mathbf{r}}{\partial \theta}$ is the local change of the boundary point with respect to θ and $\hat{\mathbf{n}}$ is the corresponding outward pointing unit normal vector. Since the boundary of A_k is the image of the boundary of U_k under ϕ (see [Section 3.3.2](#)), a parameterization can be given by

$$\mathbf{r}(t; \theta) = \phi(\gamma(t); \theta), \quad (3.17)$$

where γ is a parameterization of the square boundary of U_k . Furthermore, tangent vectors of $\mathbf{r}$ are the pushforward of tangent vectors $\dot{\gamma}$ by ϕ , i. e.,

$$\dot{\mathbf{r}}(t; \theta) = D_u \phi \dot{\gamma}(t; \theta). \quad (3.18)$$

Thus, the line element ds can be computed as

$$ds = \|\dot{\mathbf{r}}(t; \theta)\| dt \quad (3.19)$$

and the normal (up to orientation) is

$$\hat{\mathbf{n}}(t, \theta) = \frac{1}{\|\dot{\mathbf{r}}(t; \theta)\|_2} \begin{pmatrix} \dot{r}_2(t; \theta) \\ -\dot{r}_1(t; \theta) \end{pmatrix}. \quad (3.20)$$

The important quantities of the boundary integral are visualized in [Figure 3.4](#). With this, all ingredients of the derivative in [equation \(3.16\)](#) can be computed.

Note, that applying the divergence theorem on Q_{bound} would transform it into another interior integral. However, in that case a gradient computation of H with respect to the location on the sensor plane would also be required. Using the formulation above, there is no requirement on the availability of gradients of H .

3.4 IMPLEMENTATION DETAILS

In this section, some practical implementation details of the theoretical considerations above will be given. It will also be explained how the approach can fit into existing deep learning pipelines.

3.4.1 Numerical integration

The forward and backward pass are both implemented using numerical integration. Stratified Monte-Carlo integration to reduce possible aliasing artifacts that could emerge from quadrature-based integration schemes is employed. Specifically, during the backward pass, the same amount of samples on all four pixel boundaries are taken. The interior gradient integral Q_{int} looks very similar to the pixel color formula in [equation \(3.8\)](#) and can be efficiently computed by reusing samples

of $H(\mathbf{p})$ from the forward pass. The boundary integral Q_{bound} only requires sampling the border of the pixel, which requires less samples, which can be even reused for neighboring pixels, as here only the sign of the normal vector differs. No assumptions on how $H(\mathbf{p})$ is calculated are made. In general, the method can be built on top of any existing rendering algorithm, as long as it exposes a way to sample the radiance at given points on the image plane.

3.4.2 Integration into existing deep learning pipelines

The main goal of the proposed method is to enhance a downstream deep learning task, which generally requires a substantial amount of labeled training data. For many tasks, large and widely used datasets are available, whose effectiveness has been proven over time. To make use of these datasets, the radiant exposure function $H(\mathbf{p})$ is approximated with high-resolution real images, for which there are high-quality task-specific labels. These images are transformed into a coarser image using the described differentiable sensor simulator. High-resolution real images can be interpreted as samples of [equation \(3.3\)](#) on a regular grid. To sample $H(\mathbf{p})$ at arbitrary positions, bilinear interpolation is employed. Furthermore, $W_{p,k}$ is set to be constant, i. e., all points on a pixel have the same responsivity.

The differentiable sensor simulator is implemented as a custom PyTorch layer, which makes it easy to integrate into already existing projects. The layer takes a high-resolution image as input and outputs an image with the resolution of the simulated sensor. Applying tanh to the parameters restricts them to the allowed range of $(-1, 1)^2$. Since the behavior of the deformation function becomes nearly singular at the borders, the permissible range is further restricted to $(-0.6, 0.6)$. This range already allows for strong enough deformations in all experiments and also allows modeling restrictions of minimum and maximum pixel sizes in the hardware manufacturing process.

Even though non-uniform layouts are simulated, the output of the sensor layer is interpreted as lying on a uniform grid, i. e., the output is just an image tensor in $\mathbb{R}^{3 \times R_1 \times R_2}$ without information about the size and relative distance of neighboring pixels. Directly visualizing this output makes the image look deformed (see [Figure 3.2](#)). For instance, layouts with more pixels in the image center would also result in enlarged objects in the image center. This deformed image is directly fed into the downstream network. Even though this seemingly violates the implicit assumption of convolutional layers that all pixels have the same distance to their neighbors, experiments show that performance improvements over baselines are still significant. Exploring more tailored solutions that account for relative pixel positions e. g., using continuous [CNNs](#), is a subject of future work.

Implementation

3.4.3 Handling image-to-image tasks

Special care needs to be taken if the downstream network is solving an image-to-image task. In that case, there is usually a one-to-one correspondence between input pixels and output pixels, like a dense map of class probabilities for semantic segmentation. This means that a change in the pixel layout of the input image has an immediate effect on the output image, which has to be taken into account when computing the loss. The loss will, in general, be some form of distance to a reference image given in uniform pixels. A modification of the loss function properly takes the pixel area into account. To this end, a uniform grid with the reference image resolution $R_1^{\text{GT}} \times R_2^{\text{GT}}$ is warped via the inverse pixel layout deformation ϕ^{-1} , and the (deformed) output of the downstream network is sampled at these locations with nearest-neighbor interpolation. The result is an un-deformed image with constant regions that match the pixel layout defined by ϕ (see Figure 3.2). The loss of this image to the reference image can now be taken as usual. Note, that this resampling is only applied to the *output* of the downstream network, while the network itself operates on the deformed image and can thus take advantage of the non-uniform layout.

These considerations only apply to image-to-image tasks. If the output of a task does not depend on positions on the image plane (e. g. classification tasks), the loss function can be applied as-is without modifications.

3.5 EXPERIMENTS

Experiments for classification and semantic segmentation tasks were conducted. For each experiment, a task-specific network with uniform sensor layout as the baseline is compared to the same network end-to-end trained with a learned sensor layout, which is fixed during inference and evaluated on simulated data. In order to achieve the fairest possible comparisons, the uniform baselines and the learned layouts share the *exact same code* (including training schemes and hyperparameters), except that the sensor layout parameters of the baselines are frozen to 0, which results in a uniform layout.

3.5.1 MNIST classification

To illustrate the principle of the end-to-end sensor layout optimization, this section begins with a toy example that optimizes the layout for hand-written digit recognition on MNIST [Den12] with a sensor size of only 4×4 instead of the original 28×28 pixels. The digits in MNIST are always centered, so a reasonable hypothesis is that an

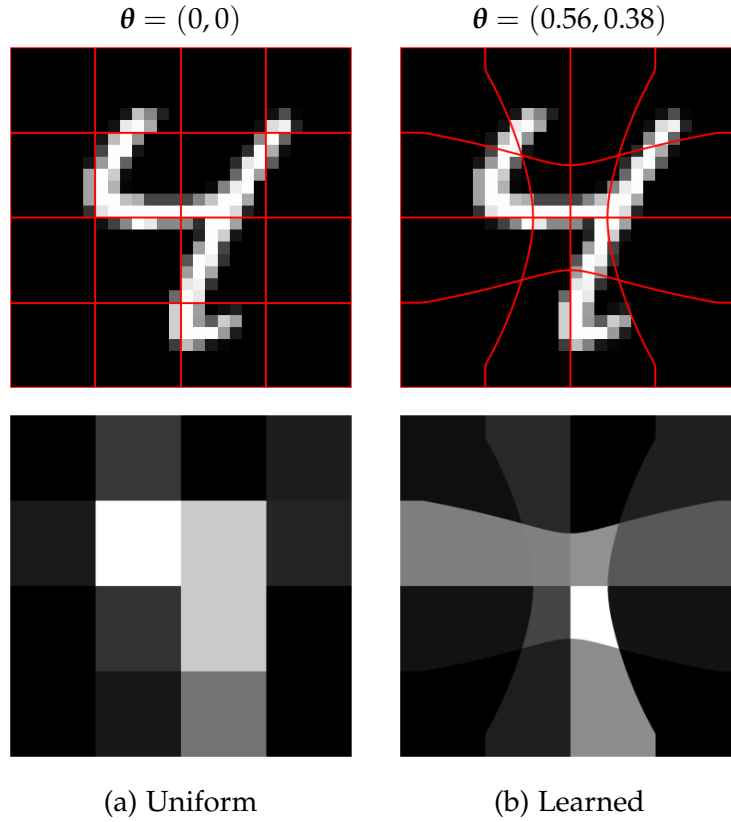


Figure 3.5: Learned sensor layout for 4×4 MNIST classification. Top row: pixel layouts overlaid over the original 28×28 image. Bottom row: Simulated sensor output. The network fine-tuned on the uniform layout wrongly classifies (a) as a 7 while the end-to-end optimized network classifies (b) as a 4.

Layout	Accuracy
Uniform	83.63%
Learned	90.90%

Table 3.1: Test set accuracy of 4×4 MNIST [Den12] classification. Even with a resolution of only 4×4 pixels, we can reach a classification accuracy of more than 90%.

optimized layout places smaller pixels in the middle to capture the higher information density there. This is in contrast to more general datasets like ImageNet [Den+09], which aim to provide images from a distribution that is as wide and general as possible, and thus have little to no spatial bias. The curvilinear layout ϕ_{curv} is applied to the sensor, with parameters initialized with 0, i.e., the training starts with the uniform pixel layout.

The output of the sensor layer is fed into a small CNN consisting of two convolutional layers with 32 and 64 channels respectively, followed by a max pooling layer and two fully connected layers with hidden dimension 128. The whole pipeline is trained for 14 epochs with Adam [KB15] and a learning rate of 0.01. The baseline is the exact same pipeline, but with frozen uniform sensor parameters.

The test set accuracy in Table 3.1 demonstrates a clear advantage of the learned layout, which can be seen in Figure 3.5. The layout confirms the hypothesis that smaller pixels in the center are advantageous for classification on this specific dataset. Note that the visualizations of the learned layout in Figure 3.5 is not what the downstream network receives as input. The CNN has no notion of pixel shape and relative distance between pixels, and only “sees” a 4×4 uniform grid of pixel values.

3.5.2 Semantic segmentation

For a relevant practical example application, the approach was also evaluated under the task of semantic segmentation of urban scenes on the Cityscapes dataset [Cor+16]. The dataset contains 5000 high-resolution, densely annotated frames with 19 object classes.

Unlike in more general segmentation tasks, street view images are taken in environments that share many similarities between individual images, and the distribution of object classes over the image plane is highly non-uniform. For instance, often large parts of the image are occupied by empty streets or the sky, while other areas have a higher density of different object classes. Thus, the segmentation task could benefit from distributing pixels to areas of higher information density.

Network	Layout	mIoU	Acc. (%)
PSPNet	Uniform	50.76	90.86
	ϕ_{curv}	52.39	91.19
	ϕ_{rect}	54.08	91.44
SegNeXt	Uniform	49.16	90.12
	ϕ_{rect}	50.01	90.22
	$^{\dagger}\phi_{\text{rect}}$	51.07	90.52
DeepLabV3plus	Uniform	55.13	91.61
	ϕ_{rect}	56.68	91.83
	$^{\dagger}\phi_{\text{rect}}$	55.95	91.79

Table 3.2: Semantic segmentation results on Cityscapes dataset [Cor+16] (256×128). Fixed layout transferred from the optimal layout of PSPNet is indicated by † . Otherwise layouts were trained from scratch. Bold indicates the best result for that network architecture.

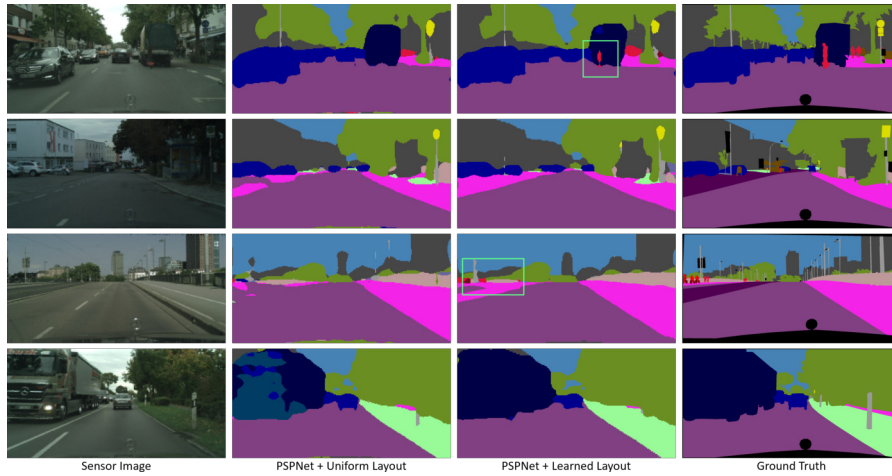


Figure 3.6: Example segmentations from the Cityscapes [Cor+16] test set with and without learned sensor layouts at 256×128 resolution. The used deformation is ϕ_{rect} . To accommodate the different pixel shapes, both variants were upsampled using nearest neighbor interpolation to the ground truth resolution (2048×1024), as described in Section 3.4.3. The learned pixel layout achieves more accurate segmentations with objects detected that were otherwise missed by using the uniform layout, especially in the vertical center of the image.

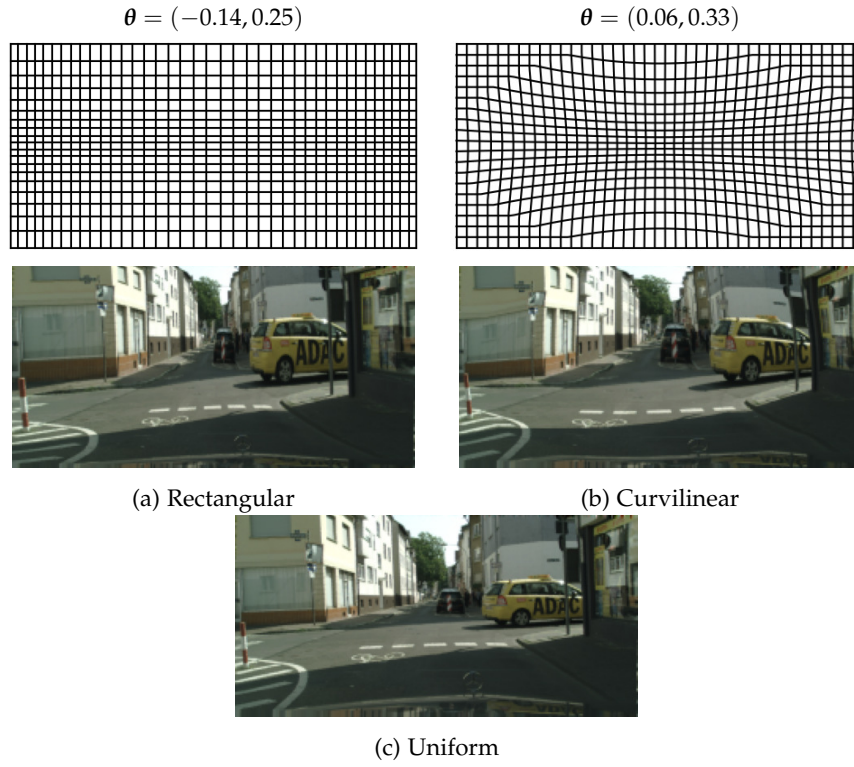


Figure 3.7: *First row:* Learned curvilinear and rectangular pixel layouts for 256×128 segmentation on Cityscapes (less pixels are shown for visualization purposes). Vertically, the sensor learned to put more pixels in the center. Further, the rectangular layout has a slightly higher pixel density towards the left and right of the sensor. *Second row:* Corresponding deformed output of the sensor simulation layer, i. e. the input to the network, for an example image. Note how areas with a higher pixel density are enlarged. *Third row:* Uniform sensor image for reference.

3.5.2.1 Baselines and setup

As baselines, the segmentation networks PSPNet [Zha+17b], SegNeXt [Guo+22] and DeepLabV3+ [Che+18] were chosen, with code taken from publicly available repositories³. Again, the baselines are trained with a fixed uniform pixel layout and compared to the same network architectures with a learnable layout. In general, segmentation networks are trained and evaluated on patches of the high-resolution input images, instead of downsampling them. Since the objective is to optimize the pixel layout over the whole sensor plane, here the whole input is fed to the network directly instead. This also means that not all data augmentation schemes from [Zha+17b] can be employed, namely random rotation, which were disabled for the experiments. All sensor parameters are initialized with 0 (uniform layout) and frozen for 10 epochs to speed up convergence to reasonable network weights. Apart from that, the same default hyperparameters were used for baseline experiments and experiments with learned layouts.

To assess whether curvilinear or rectangular layouts have an advantage over the other, both variants were trained. In order to evaluate whether a layout optimized for one network can be transferred to another on the same task, the optimal rectangular layout from PSPNet was transferred to the other networks. For these experiments, the transferred PSPNet layouts were frozen, while the networks were trained from scratch. Moreover, PSPNet is evaluated on three different sensor resolutions.

3.5.2.2 Results

The learned layouts in Figure 3.7 display a higher pixel density toward the horizon line. As expected, fewer pixels are needed to detect the street and the sky. Perspective also dictates that objects far away from the camera gather at the horizon and are smaller, thus requiring a finer pixel raster to distinguish them. Interestingly, the sensor with the rectangular layout also learned to put slightly more pixels toward the left and right edges, leading to two distinct high pixel density areas. This might be due to the fact that there is often a higher density of small objects on the sidewalks (people, poles, fences, vegetation) as opposed to the street itself. The learned curvilinear layout does not show this horizontal non-uniformity. Although hard to verify, an explanation for this could be that the deformation here is restricted to a circular region, which means that the region at the sensor boundary cannot be sampled more densely.

The quantitative results in Table 3.2 and Table 3.3 show a clear advantage over the uniform layout on all resolutions tested. In all experiments, the rectilinear layout outperformed their curvilinear

³ PSPNet experiments are based on <https://github.com/hszhaeo/semseg> and the other two networks on <https://github.com/open-mmlab/mmdetection>

Resolution	Method	mIoU (%)	Acc. (%)
128×64	PSPNet [Zha+17b]	37.11	87.17
	PSPNet + ϕ_{curv}	37.56	87.33
	PSPNet + ϕ_{rect}	39.44	87.90
256×128	PSPNet	50.76	90.86
	PSPNet + ϕ_{curv}	52.39	91.19
	PSPNet + ϕ_{rect}	54.08	91.44
512×256	PSPNet	63.98	93.73
	PSPNet + ϕ_{curv}	64.65	93.73
	PSPNet + ϕ_{rect}	64.74	93.75

Table 3.3: Ablation of semantic segmentation results of PSPNet with different sensor resolutions on the Cityscapes dataset [Cor+16]. For all resolutions, the learned rectangular layout performs best.

counterparts, which might be because of their limited adaptability in the image corners. The layouts transferred from PSPNet also achieve improved results, which indicates that the layouts are task-specific rather than network-specific. Figure 3.6 also show encouraging results for the detection of objects in the image center.

The more recent architectures DeepLabV3plus and SegNeXt seem to gain less improvement from the learned layouts. One possible explanation is that their higher representational capacity allows them to compensate for the limitations of the uniform grid. Another possible explanation is that these specialized architectures are less effective in processing deformed inputs, thus diminishing the benefit of the learned layouts. An analysis of this behavior and the development of specialized architectures tailored for the deformed inputs remains an interesting direction for future work.

3.5.2.3 Accuracy gain in different sensor areas

To show the influence of pixel density on per-pixel accuracy, the accuracy gain of the learned layouts compared to the uniform baseline are compared by computing the mean accuracy at all pixel positions. Figure 3.8 exemplary shows the accuracy gain of the learned layouts with the PSPNet downstream network. It can be seen that the accuracy significantly increases in regions with higher pixel density, while staying the same or only slightly decreasing in low density regions. This is evidence that the learned layout successfully puts higher density into regions that are otherwise harder for the network to discriminate while putting less emphasis into easier areas (like the street). Furthermore, the area of accuracy increase over the uniform layout seems to broaden in vertical direction toward the left and right corners

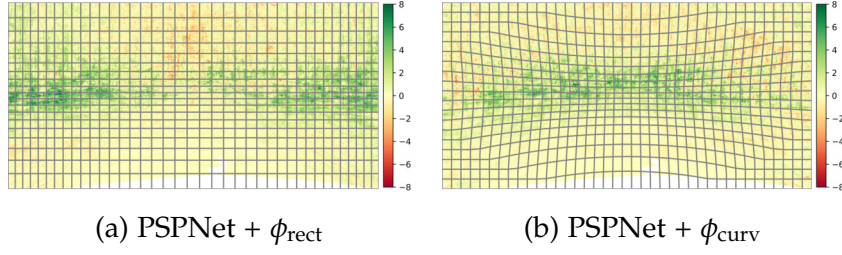


Figure 3.8: Difference of per pixel accuracy in percentage points compared to the uniform layout for 256×128 PSPNet on Cityscapes. Values greater than 0 indicate image regions where the learned layout performs better. The per pixel accuracies were computed as the average accuracy over the whole test set. The corresponding layouts are overlaid and do not represent the actual resolution of 256×128 for visualization purposes.

Resolution	Method	Accuracy	θ
8×8	ResNet18	86.60%	-
	ResNet18 + ϕ_{rect}	87.19%	(0.32, 0.08)
16×16	ResNet18	88.39%	-
	ResNet18 + ϕ_{rect}	88.62%	(0.15, 0.03)

Table 3.4: Accuracy and pixel layouts for classification on the CelebA dataset [Liu+15].

for the rectangular layout. Apparently, the current curvilinear layout parameterization cannot properly cope with the requirement of having higher pixel resolution in the left and right image centers. In future work alternative curvilinear parameterizations could be investigated.

3.5.3 Multi-label classification

Finally, the method is evaluated on the multi-label classification of facial attributes on the CelebA dataset [Liu+15]. The dataset contains 200K frontal images of celebrities and is annotated with 40 facial attributes. Similarly to MNIST, the faces are in the image center, so that a non-uniform pixel layout could be advantageous. A ResNet18 [He+16] classification network trained on ImageNet [Den+09] is fine-tuned on CelebA over 15 epochs on different resolutions using the rectilinear pixel layout. The results in Table 3.4 show marginal improvements over the baselines. As expected, the network learns a higher pixel density in the image center. The joint prediction of 40 different attributes (spatially spread over each face) can be the reason for the comparatively small improvements.

3.6 CONCLUSIONS AND LIMITATIONS

This chapter presented the first approach of a parameterizable and differentiable pixel layout that can be jointly optimized with any downstream, task-specific network in an end-to-end fashion. This approach allows going beyond fixed, regular pixel layouts that treat all regions of an image as equally important, yielding task-specific layouts on which the networks perform better than on regular grids. The chapter provided the generic concept of an analytic, differentiable sensor layout parameterization that retains a regular topology and presented two pixel layout parameterizations, i. e., rectangular and curvilinear grid shapes. The drop-in module allows to apply the learnable pixel layout to existing high-resolution imagery, and, thus, connecting the proposed method to existing deep learning pipelines. Experiments showed that network predictions benefit from a learnable pixel layout for classification and semantic segmentation tasks.

The proposed deformation functions are comparatively simple, do not cover all task preferences and require more investigation. In addition, not all tasks have a sufficient spatial bias, i. e., a non-uniform distribution of objects of interest in the image plane to make non-uniform sensor layouts advantageous. Yet, sufficiently many application scenarios still naturally benefit from non-uniform layouts. Moreover, a tighter integration of the sensor design parameters and their (geometric) limits into the modeling of deformations is needed. In particular, the proposed model assumes a 100% fill factor, whereas practical hardware implementations are subject to constraints such as finite fill factors, placement of pixel electronics, and other fabrication limitations. This is particularly important for future work on the hardware design and optimization of curvilinear and more free-form layouts.

Limitations

4

LEARNING COLOR FILTERS FOR DEEP LEARNING TASKS

While the previous chapter described a way to optimize the physical location and shape of pixels on a sensor plane to improve various tasks, this chapter evaluates whether *color filters* tailored specifically for semantic segmentation tasks solved with deep learning methods can have an advantage over conventional hand-crafted filters. To this end, an approach to optimize color filters jointly with neural network parameters is presented. In contrast to previous work, this method also incorporates a realistic shot noise model into the optimization, which is crucial, since color filters directly influence the signal-to-noise ratio of the acquired image. However, under these realistic conditions, an extended evaluation suggests no statistically significant or robust advantage of learned filters over hand-crafted baselines on the task of semantic segmentation for autonomous driving.

4.1 INTRODUCTION

With the advent of deep learning, neural networks have revolutionized many aspects of image data processing by solving imaging applications in a data-driven fashion. While most applications of neural networks in image processing implicitly assume a completely separate upstream image formation and acquisition process, and thus do not incorporate sensor design parameters in the learning process, recent work showed that optimizing parameters such as the position and shape of pixels [Cho+24; Som+24; KN25] (see Chapter 3), various optical components [Cha+18; CW19; Chu+21; Met+20; Sun+20; Tse+21], or parts of the image signal processor (ISP) pipeline [Dia+21; Mos+20] together with neural networks can improve task-specific performance.

Other works have also proposed optimizing color filters for the tasks of demosaicing with neural networks [Cha16; HGO18], i. e., optimizing the CFAs as described in Section 2.1.2.3 themselves, but also training a neural network that predicts the missing channels at each pixel. These works highlight the potential of integrating hardware-level optimizations with algorithmic advancements, suggesting that a more holistic approach can yield significant improvements in image quality and processing efficiency. However, the objective in filter design for demosaicing is to find the best color filters for reconstructing images *optimized for human perception*. While the learned filters improve the quality of the input image and thus can also increase the performance of general downstream tasks, they may not be optimal for more spe-

Motivation

cific scenarios like low-light imaging, or higher-level computer vision tasks like semantic segmentation, where finer nuances in the color spectrum than the human eye is able to perceive could be beneficial to separate different classes of objects. This chapter investigates whether similar gains can be realized for a high-level task such as semantic segmentation. It aims to explore the intersection of deep learning and spectral color filter optimization, presenting a framework to optimize both neural networks and color filters in tandem to critically assess whether such performance gains exist under realistic conditions. The main contributions in this chapter are

Contributions

- an end-to-end differentiable drop-in module that realistically simulates color filtering in sensors, including shot noise, and enables learning of physical filter response functions for any downstream task, such as semantic segmentation, and
- a systematic evaluation showing that, under realistic noise and training conditions, learned filters do *not* reliably outperform strong hand-crafted baselines, and an analysis of possible reasons for these results.

4.2 RELATED WORK

In terms of color filter design for demosaicing, Chakrabarti [Cha16] learns an optimal CFA pattern by choosing filters from a set of predefined filters (red, green, blue and white) in a 3×3 area in a differentiable fashion, by gradually increasing the temperature parameter of a softmax layer. In contrast, Henz et al. [HGO18] directly learn the CFA filter weights. Both works simulate the color filters with existing 3-channel RGB images, which means that no spectral response curves can be learned. Since the input images only have 3 channels already, no new spectral information can be gained by the learned filters. Additionally, both works do not consider shot noise in their sensor models. In comparison, the approach presented in this chapter includes a sensor simulation with a differentiable shot noise model, and utilizes hyperspectral images to optimize the full spectral response curves.

Demosaicing

Another related research area is band selection to reduce the amount of information in hyperspectral imaging, either for spectral reconstruction [LDV23; Fu+20], or for efficiently solving a deep learning task [Mic+20; Mou+21]. Band selection selects single channels of a hyperspectral image, while the method presented in this chapter aims to design physically realizable filter response curves that do not require a hyperspectral camera at inference time.

Band selection

The most closely related work to the approach presented here is [Nie+18]. The authors also end-to-end optimize filter response curves from hyperspectral images, but only for the task of spectral reconstruc-

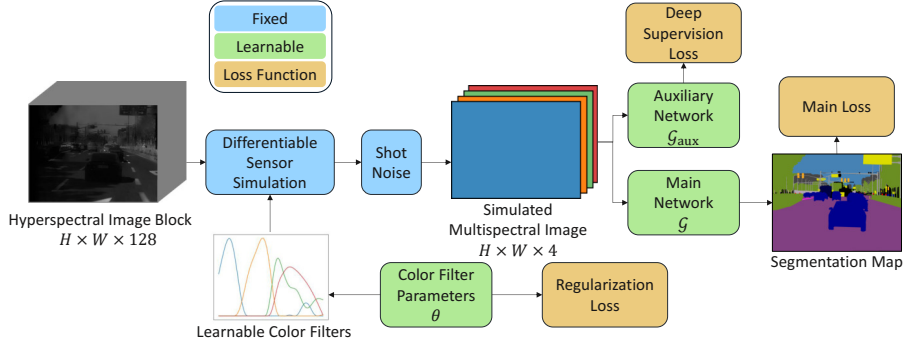


Figure 4.1: Overview of the training pipeline. The method utilizes hyperspectral data to simulate multispectral images with learnable filter curves and passes the resulting simulated image to the downstream segmentation network. Additionally, the filter curves are regularized with Laplacian and deep supervision loss terms, which utilize a very shallow auxiliary network, that is trained together with the main network.

tion, and not for arbitrary high-level vision tasks. Moreover, they do not simulate shot noise in their sensor model, while the approach in this chapter explicitly includes differentiable shot noise, which is crucial for an accurate sensor simulation, as the shape of filter response curves influences the signal-to-noise ratio of the intensities measured by each pixel, as explained in [Section 2.1.2.5](#). To the best of current knowledge, no prior work has systematically evaluated whether such learned spectral filters improve high-level semantic segmentation tasks under a realistic sensor noise model. The experiments in this chapter address this gap and find no consistent advantage over hand-crafted filters.

4.3 METHOD

This section will start with a brief introduction to the overall approach and how it fits into general deep learning pipelines. The following sections will then go into more detail.

Like in [Chapter 3](#), images are not considered fixed, but functions that depend on sensor parameters. The approach presented here considers the (multispectral) images to be functions $I_{\text{ms}}(I_{\text{hs}}; \theta)$, that map learnable color filter parameters θ and a hyperspectral image (HSI) I_{hs} to a C -channel multispectral image $I_{\text{ms}} \in \mathbb{R}^{H \times W \times C}$. This image can then be used as input to a task-specific downstream network $\mathcal{G}(\cdot, \nu)$ with learnable network parameters ν . If the image formation process is differentiable with respect to the sensor parameters θ , they

Training process

can be *jointly* learned together with the network parameters ν by optimizing

$$\min_{\theta, \nu} \mathbb{E}_{(I_{hs}, \mathbf{y})} (\mathcal{L}(\mathcal{G}(I_{ms}(I_{hs}; \theta), \nu), \mathbf{y})), \quad (4.1)$$

where $\mathcal{L}$ is a loss function and $\mathbf{y}$ is the ground truth label corresponding to the hyperspectral image I_{hs} inside the training data set. Notably, this only describes the training process, and during inference the hyperspectral input images are no longer needed, as I_{ms} can be directly measured by a physical realization of the optimized spectral filters. The following sections will outline the differentiable image formation process, i. e., how $I_{ms}(I_{hs}; \theta)$ is computed. For an overview of the overall method see [Figure 4.1](#).

4.3.1 Image formation model

The measurement model from [\[Nie+18\]](#) is used in the proposed method, which in the following section will be derived from the more general model in [equation \(2.25\)](#). The basis for the sensor simulation pipeline is the general measurement equation of a single pixel with index $k = (i, j)$ as described in [Section 2.1.2](#). A multi-chip setting like in [\[Nie+18\]](#) is assumed, i. e., each pixel has multiple color filters, without the need of demosaicing. This also means that every pixel has multiple weighting functions $W_{k,c}$, where c denotes the channel. Under the light assumption that that the weighting function can be separated as $W_{k,c}(\mathbf{p}, \omega, t, \lambda) = S_c(\lambda) \tilde{W}_k(\mathbf{p}, \omega, t)$, i. e., that color filters $S_{k,c}$ are independent of the position *inside* the pixel, the incident ray direction and time, integrating [equation \(2.25\)](#) over area, hemisphere and exposure time yields

$$\begin{aligned} \bar{I}_{ms,k,c} &= \int_{\lambda_{\min}}^{\lambda_{\max}} \int_{t_{\min}}^{t_{\max}} \int_{A_k} \int_{\Omega} W_{k,c}(\mathbf{p}, \omega, t, \lambda) L_{e,\lambda}(\mathbf{p}, \omega, t, \lambda) d\omega d\mathbf{p} dt d\lambda \\ &= \int_{\lambda_{\min}}^{\lambda_{\max}} S_c(\lambda) I_k(\lambda) d\lambda, \end{aligned} \quad (4.2)$$

with

$$I_k(\lambda) = \int_{t_{\min}}^{t_{\max}} \int_{A_k} \int_{\Omega} \tilde{W}_k(\mathbf{p}, \omega, t) L_{e,\lambda}(\mathbf{p}, \omega, t, \lambda) d\omega d\mathbf{p} dt. \quad (4.3)$$

Every pixel k and channel c of a hyperspectral image I_{hs} can be approximated as the integral of $I_k(\lambda)$ over a narrow wavelength interval. Thus, discretizing [equation \(4.2\)](#) and parameterizing S_c with learnable parameters θ , which model the shape of the color filters, yields

$$\bar{I}_{ms,k,c}(\theta) = \sum_i S_c(\lambda_i; \theta) I_{hs,k}(\lambda_i), \quad (4.4)$$

which can be directly computed from a hyperspectral image $I_{hs,k}$ [\[Nie+18\]](#).

Hyperspectral image

Like [Nie+18], an individual trainable parameter $\theta_{i,c}$ for each discretized wavelength λ_i is introduced, so the computation of final intensities reduces to a simple dot product of the weight vector and the input HSI pixel, i. e.

$$\bar{I}_{ms,k,c}(\boldsymbol{\theta}) = \langle \boldsymbol{\theta}_c, \mathbf{I}_{hs,k} \rangle. \quad (4.5)$$

This function is trivially differentiable with respect to $\boldsymbol{\theta}$, enabling gradient-based optimization schemes. Physically valid color filters require the components of $\boldsymbol{\theta}$ to be bounded between 0 and 1. This property is achieved by clipping the weights after each optimization step.

4.3.2 Differentiable shot noise

As color filters indirectly influence the signal-to-noise ratio of the final image (see Section 2.1.2.5), in contrast to prior work [Nie+18], the proposed method assumes that the measurements follow a Poisson distribution. In order to simulate different light levels and exposure times, a fixed parameter α similar to [Dia+21] is introduced. Denoting a random number drawn from a Poisson distribution with parameter x as $\mathcal{P}(x)$, the final image intensities are given by

$$I_{ms,k,c} = \max(\mathcal{P}(\alpha \bar{I}_{ms,k,c}) / \alpha, 0), \quad (4.6)$$

where the expected intensity $\bar{I}_{ms,k,c}$ is computed as in equation (4.5). The parameter α is used to model multiple sensor and scene parameters that influence shot noise strength, like exposure time and light level, while keeping the dynamic range fixed.

Simulating light levels

Since sampling from a Poisson distribution is not directly differentiable, a Gaussian approximation and the reparameterization trick [Kin13] are utilized in order to detach the random sampling process from the computation graph:

Reparameterization trick

$$\mathcal{P}(x) \approx x + \sqrt{x} \mathcal{N}(0, 1), \quad (4.7)$$

where $\mathcal{N}(0, 1)$ is a random number drawn from a standard normal distribution. This approximation is accurate for sufficiently large light levels or exposure time, and is trivially differentiable with respect to x , which enables the propagation of gradients all the way through the downstream network and the image formation process back to the color filter parameters.

Including shot noise in the sensor model is important for an accurate filter simulation, as wider filter curves lead to a higher signal-to-noise ratio, while narrower curves potentially increase the discriminative potential of individual channels. The differential shot noise model allows the optimization process to account for this trade-off between noise robustness and spectral selectivity.

4.3.3 Loss functions

The main loss function is a task-specific objective function (e. g., cross-entropy loss for semantic segmentation)

$$\mathcal{L}_{\text{main}}(\mathbf{I}_{\text{hs}}, \mathbf{y}) = \mathcal{L}_{\text{task}}(\mathcal{G}(\mathbf{I}_{\text{ms}}(\mathbf{I}_{\text{hs}}; \boldsymbol{\theta}); \mathbf{v}), \mathbf{y}). \quad (4.8)$$

Optimizing the per-wavelength weights $\boldsymbol{\theta}$ completely independently leads to very chaotic filter response curves, which would be hard to manufacture. For this reason, a Laplacian regularization term is introduced in addition to the usual loss function used for the specific downstream network:

Filter regularization

$$\mathcal{L}_{\text{reg}} = \sum_c \sum_i (\theta_{i-1,c} - 2\theta_{i,c} + \theta_{i+1,c})^2. \quad (4.9)$$

The term penalizes steep edges in the filter response curves and leads to smoother filters.

Additionally, for very deep networks, gradients propagated to the input layer can become very small. In order to improve the gradient flow, an approach similar to deep supervision [Lee+15] is introduced. To this end, a very shallow network $\mathcal{G}_{\text{aux}}$ is added. This network solves the same task as the downstream network, while sharing the same color filter weights, effectively forking the output of the sensor simulation to the two networks and training them simultaneously end-to-end. Note that this network is only required during the training process and discarded afterwards. The weights of the auxiliary network are denoted with $\mathbf{v}_{\text{aux}}$. The deep supervision loss term is analogously defined to the main loss:

Deep supervision

$$\mathcal{L}_{\text{ds}}(\mathbf{I}_{\text{hs}}, \mathbf{y}) = \mathcal{L}_{\text{task}}(\mathcal{G}_{\text{aux}}(\mathbf{I}_{\text{ms}}(\mathbf{I}_{\text{hs}}; \boldsymbol{\theta}); \mathbf{v}_{\text{aux}}), \mathbf{y}). \quad (4.10)$$

Finally, the main loss, the Laplacian regularization term, and the deep supervision loss are combined in a single loss function

$$\mathcal{L}_{\text{tot}}(\mathbf{I}_{\text{hs}}, \mathbf{y}) = \mathcal{L}_{\text{main}}(\mathbf{I}_{\text{hs}}, \mathbf{y}) + \eta_{\text{ds}} \mathcal{L}_{\text{ds}}(\mathbf{I}_{\text{hs}}, \mathbf{y}) + \eta_{\text{reg}} \mathcal{L}_{\text{reg}}, \quad (4.11)$$

where $\eta_{\text{ds}}, \eta_{\text{reg}} \geq 0$ are weighting hyperparameters.

4.4 EXPERIMENTS

This section investigates the effectiveness of the proposed method for semantic segmentation in autonomous driving.

4.4.1 Experiment setup

Experiments are conducted on the HSICityV2 dataset [Hua+21], which contains 1256 hyperspectral street view images with coarse ground

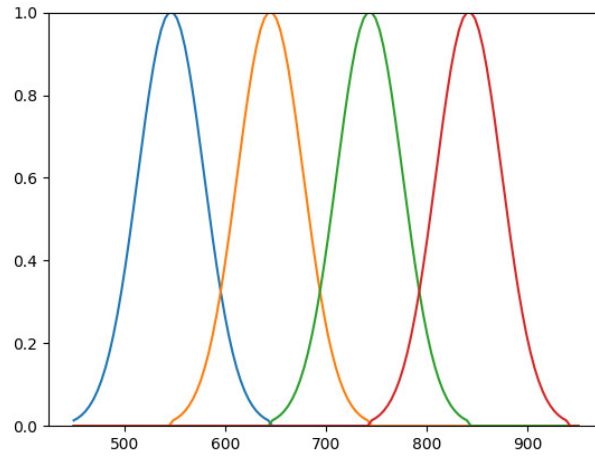


Figure 4.2: Baseline color filters with 4 channels. The learned filters are also initialized with these curves.

truth semantic segmentation masks with 19 classes. The images contain 128 equidistant wavelength bands in the visible and near-infrared spectrum (450 nm to 950 nm). This fine spectral resolution enables an accurate color filter simulation. The data is split into 1030 training images and 226 test images.

The approach is evaluated on two downstream network models: the energy efficient MobileNetV3 [How+19] with an LR-ASPP [Che+17] head and the lightweight Fast-SCNN architecture [PLC19]. Both networks were designed for energy efficiency and fast inference. The baselines for all experiments are the hand-crafted multispectral color filters from [BAo6]. The filters consist of Gaussians that are evenly placed over the input spectrum; see Figure 4.2. The learnable filters of the proposed method are initialized with the same filter curves as the baseline. A shallow CNN with only a single hidden layer with 32 channels and kernel size 3 is used.

Networks

The influence of different choices of hyperparameters, namely the number of channels, regularization weights η_{reg} and deep supervision weight η_{ds} , is also evaluated. For the fairest possible comparison, both the baseline experiments and the corresponding experiments employing the proposed method use the exact same hyperparameters in all cases, and no changes in the training processes were made. All networks are trained from scratch for each experiment for the same number of iterations.

Hyperparameters

The implementation is based on the publicly available *mmsegmentation* repository [MMS20] and the default configurations are used for the MobileNetV3 and Fast-SCNN experiments. The experiments include the commonly used evaluation metrics for semantic segmenta-

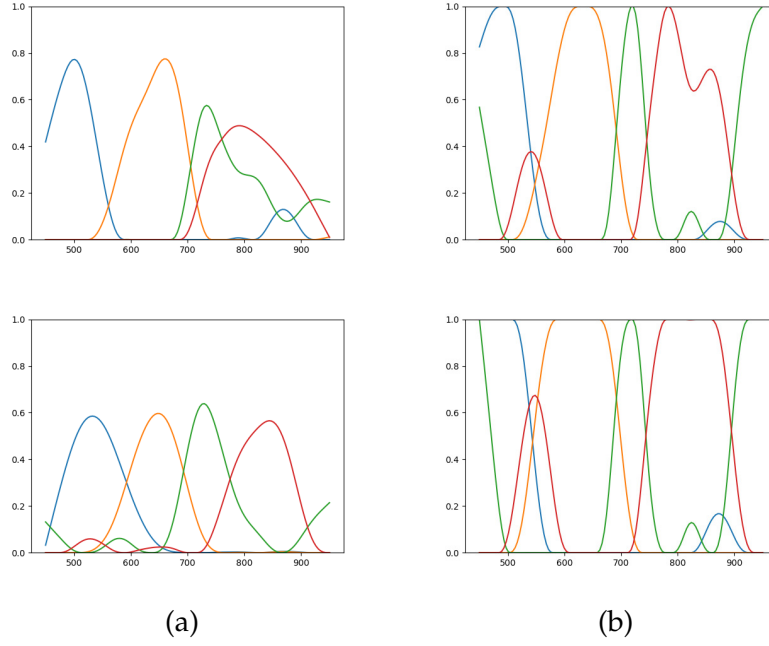


Figure 4.3: Learned filters for MobileNet (top row) and Fast-SCNN (bottom row). (a) without deep supervision (b) with $\eta_{ds} = 1$. All curves used the standard regularization weight $\eta_{reg} = 0.1$

tion, namely mean accuracy (mAcc) and mean intersection over union (mIoU).

4.4.2 Results

The test set results for 4-channel multispectral color filters for different choices of hyperparameters are in [Table 4.1](#). The optimized filter curves for some experiments are shown in [Figure 4.3](#). The use of deep supervision drastically changes the learned curves; see [Figure 4.3](#). Without deep supervision (left column), the filters do not have maximums at 1, and thus attenuate light at every wavelength. This is counterproductive because less light hitting the sensor leads to a worse SNR ([Section 2.1.2.5](#)). A possible explanation for this effect are the weak, vanishing gradients that are propagated through the deep network. Since the task-based loss gradient is small, the gradient of the regularization term dominates. The right column of [Figure 4.3](#) shows that this can be effectively combated with deep supervision. The task-based gradients propagated through the shallow network are larger and help the filters improve the signal-to-noise ratio themselves, resulting in curves that make use of the whole spectrum and not unnecessarily attenuate all wavelengths.

However, despite the very different filter shapes, the results in terms of mAcc and mIoU in [Table 4.1](#) barely differ for different values of η_{ds} . This suggests that the fluctuations in performance are dominated

Method	η_{reg}	η_{ds}	mAcc	mIoU
Fast-SCNN (fixed)	-	-	46.69	35.37
Fast-SCNN	0.1	-	47.30	36.77
Fast-SCNN	0.1	0.4	46.79	36.53
Fast-SCNN	0.1	1.0	47.86	<u>36.75</u>
Fast-SCNN	0.1	2.0	<u>47.37</u>	36.48
Fast-SCNN	0.05	0.4	46.47	35.92
MobileNetV3 (fixed)	-	-	60.59	47.67
MobileNetV3	0.1	-	60.62	44.99
MobileNetV3	0.1	0.4	<u>60.77</u>	47.82
MobileNetV3	0.1	1.0	60.76	<u>47.92</u>
MobileNetV3	0.1	2.0	61.13	45.58
MobileNetV3	0.05	0.4	60.25	47.73

Table 4.1: Segmentation Results With 4 Channels for different hyperparameters

Run	Fixed Filters	Learned Filters
#1	60.4/45.17	59.93/45.14
#2	56.4/44.30	56.79/43.87
#3	60.19/47.16	60.29/47.22
#4	60.65/47.8	60.68/47.13
#5	60.42/47.67	60.76/47.92
Avg.	59.61/46.42	59.69/46.26
Std.	1.80/1.59	1.66/1.69

Table 4.2: mAcc/mIoU of repeated experiments with MobileNetV3 ($\eta_{\text{reg}} = 0.1$ and $\eta_{\text{ds}} = 1.0$). For every run, the learned and fixed filter variants were trained from scratch with the same random seed.

by stochasticity in the training process (e.g., random initialization of network weights and batch selection) rather than by the specific learned filter curves. To validate this hypothesis, the MobileNetv3 experiment was repeated multiple times with different random seeds. In each run, the baseline experiments with fixed filters and the corresponding experiment with learned filters used the same random seed, such that the networks were initialized with the same weights and no other differences in the training process were introduced. The results in [Table 4.2](#) confirm that the learned filters have less influence than the randomness in the training process. For a given seed, there is barely a difference between using learned and fixed filters, and in some runs the fixed filters even slightly outperform the learned filters. The mean differences between the two variants are also much smaller than the corresponding standard deviations, indicating that no robust performance advantage of the learned filters can be observed.

4.4.3 Discussion

The results in [Tables 4.1](#) and [4.2](#) indicate that jointly learned color filters do not provide a statistically significant advantage over the hand-crafted baselines for semantic segmentation in the considered setting, i.e., autonomous driving. A possible explanation is that, in the task of autonomous driving under outdoor illumination, the discrimination between classes is dominated by texture and geometric structure, i.e., object shape and surface appearance, so only very little additional benefit can be gained from fine-grained spectral information. The color filters implement a linear spectral mixing of the hyperspectral data into a lower-dimensional vector at each pixel. Many different choices of filters may be functionally equivalent for solving the final task. The deep downstream networks have a high capacity and can adapt to a large amount of different spectral filters, thus reducing the potential benefit that an optimized color filter could provide.

The comparatively stronger gains achieved by learning pixel layouts (see [Chapter 3](#)) support this analysis, as higher pixel densities in important areas further increase the ability of the network to detect different shapes and textures. This suggests that, at least for the evaluated segmentation task with street-view images, the spatial distribution of pixels has a larger influence on the downstream task performance than tailored spectral filters. However, this conclusion is only valid for the setting in this chapter. Other tasks or data may benefit more from color- or spectrum-based class discrimination.

4.5 CONCLUSIONS AND FUTURE WORK

This chapter presented an end-to-end framework to optimize color filter curves for imaging sensors together with a neural network with the goal to solve a high-level downstream task. The method consists of a realistic sensor simulation from hyperspectral images, including a differential shot noise model, a parameterization of spectral filter curves, and regularization and deep supervision schemes to obtain physically plausible filters.

Under realistic noise conditions and in the context of semantic segmentation for autonomous driving, the experiments did not reveal a significant advantage of jointly learned color filters over hand-crafted baselines. The performance difference between the learned and fixed filters was consistently lower than the variability due to randomness in the training process, i. e., weight initialization and batch sampling, suggesting that downstream networks rely more on shape and texture than spectral information.

Despite the negative results for the specific tasks considered in this chapter, the framework can still be applied to other downstream tasks and datasets. Its use, however, is currently constrained by the availability of large-scale [HSI](#) datasets with task-specific labeling, which are crucial for the training of the deep downstream networks.

There are several possible directions for future work. Tasks or domains that rely more heavily on spectral information, such as remote sensing or material classification, may benefit more from learned filters than the autonomous driving scenario investigated in this chapter. Moreover, a more explicit integration of color filter manufacturing constraints into the optimization process could be explored. Future work could also aim to automatically find the optimal number of channels for a given task and budget, rather than treating the channel count as a fixed hyperparameter. This could be especially interesting in the mosaiced single-chip setting, since the number of distinct filters determines the spatial arrangement and distance of corresponding filters on the sensor.

Future work

5

ENERGY DISSIPATION WITH PLUG-AND-PLAY PRIORS

After examining methods for improving imaging pipelines at the sensor level in the previous chapters, this chapter aims to improve already captured images in a provable manner. Neural networks have reached outstanding performance for solving various ill-posed inverse problems in imaging. However, challenges of end-to-end learning approaches in comparison to classical variational methods are the requirement of expensive retraining or finetuning for even slightly different problem statements and the lack of provable error bounds during inference. Prior methods tackled the first problem by using networks trained for Gaussian image denoising as generic plug-and-play regularizers in energy minimization algorithms. Even though this obtains state-of-the-art results on many tasks, heavy restrictions on the network architecture have to be made if provable convergence of the underlying fixed point iteration is a requirement.

Other work has proposed to train networks to output descent directions with respect to a given energy function with a provable convergence to a minimizer of that energy. However, each problem and energy requires the training of a separate network. This chapter considers a combination of both approaches by projecting the output of a plug-and-play denoising network onto the cone of descent directions of a given energy function. This way, a single pretrained denoising network can be used for a wide variety of reconstruction tasks. The results show improvements compared to classical energy minimization methods while still having provable convergence guarantees. The approach developed in this chapter was published in [SKM19].

5.1 INTRODUCTION

As outlined in [Section 2.3](#), in many image processing tasks an observed image f is modeled as the result of the transformation of a clean image $\hat{u}$ under a known linear transformation A and unknown (Gaussian) noise ξ , i. e.,

$$f = A\hat{u} + \xi. \quad (5.1)$$

In most cases, the problem of reconstructing $\hat{u}$ from f and A is ill-posed and can thus not be solved by a simple inversion of A , giving rise to the field of regularization theory with iterative or variational methods, see e. g. [BB18] for an overview. In recent years, neural networks were very successful in learning a direct mapping $\mathcal{G}(f; \theta) \approx \hat{u}$

for a variety of problems such as deblurring [Xu+14; Tao+18], denoising [Zha+17a], super-resolution [Don+15], demosaicing [Gha+16] and MRI- or CT-reconstruction [Yan+17; KMY17]. Even though this works well in practice, there are rarely any guarantees on the behavior of neural networks on unseen data, making them difficult to use in safety-critical applications. Moreover, for each type of problem a separate network has to be trained.

The classical variational method outlined in Section 2.3.1 is an alternative approach that tries to find the solution by minimization of a suitable energy function

$$\hat{u} = \operatorname{argmin}_u H_f(u) + \alpha R(u), \quad (5.2)$$

with data fidelity term H_f and regularization function R . Although minimizers of this kind come with many desirable theoretical guarantees, common regularizations such as TV often cannot perfectly capture the complex structure of the space of natural images.

A particularly flexible approach arises from the hybrid plug-and-play methods in Section 2.3.2 in which the proximal operator of the regularizer is replaced by an arbitrary denoising networks. While such approaches allow tackling different inverse problems with the same neural network, the derivation of theoretical guarantees - even in terms of convergence of the resulting algorithmic scheme - remains difficult, see [CWE16; Sre+16] or some discussion in [MC18], unless the denoiser satisfies particular properties [Ryu+19; REM17]. The contribution of this chapter is a *provably convergent* plug-and-play algorithmic scheme, which makes no restrictions on the used denoising network.

5.2 PROVABLY DISSIPATING MODEL-BASED ENERGIES WITH PLUG-AND-PLAY PRIORS

The starting point of the above-mentioned algorithmic schemes that utilize denoising networks to regularize model-based inverse problems are the (proximal) energy minimization algorithms from Section 2.3.1.2. While most works focus on primal-dual / ADMM approaches, their convergence analysis is quite delicate, even in a setting, in which one still minimizes (nonconvex) energies, so this chapter turns to two simpler methods, namely gradient descent and proximal gradient methods, whose algorithmic schemes are repeated here:

$$u_{t+1} = u_t - \tau \left(A^\top \nabla H_f(Au_t) + \alpha \nabla R(u_t) \right), \quad \text{and} \quad (5.3)$$

$$u_{t+1} = \operatorname{prox}_{\tau R} \left(u_t - \tau A^\top \nabla H_f(Au_t) \right). \quad (5.4)$$

Following the idea of [Hei+14; VBW13; DKE10], considering either a gradient descent or a proximal step on the regularization as a

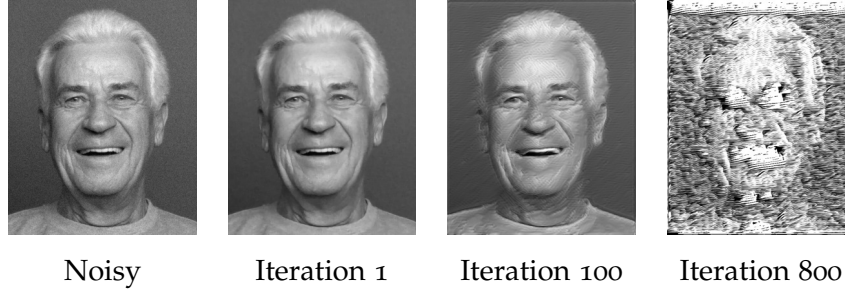


Figure 5.1: Example of a diverging algorithmic scheme using [equation \(5.5\)](#) with $H_f = 0$ and a DnCNN denoising network $\mathcal{D}$. Pixel values are clipped to $[0, 1]$ for visualization. Images are taken from the FACES dataset¹.

generic denoising operation gives rise to the following two algorithmic schemes,

$$u_{t+1} = \frac{1}{2}\rho(u_t; 2\tau) + \frac{1}{2}\mathcal{D}(u_t), \quad (5.5)$$

$$u_{t+1} = \mathcal{D}(\rho(u_t; \tau)), \quad (5.6)$$

where $\mathcal{D}$ denotes any kind of denoiser, e. g., a convolutional neural network, and a gradient descent step on the data term is shortened to $\rho(u_t; \tau) = u_t - \tau A^\top \nabla H_f(Au_t)$ for brevity of notation.

Algorithmic schemes like [equations \(5.5\)](#) and [\(5.6\)](#) combine the model-based flexibility of energy minimization methods, i. e., the explicit modeling of H_f , with the expressive power of deep neural networks. Unfortunately, despite their success in various practical applications, schemes like [equations \(5.5\)](#) and [\(5.6\)](#) remain dangerous to be used: [Figure 5.1](#) shows the result of running the iteration [equation \(5.6\)](#) with $H_f = 0$ on a noisy input image $f = u_0$ for 100 and 800 iterations using a DnCNN [[Zha+17a](#)] denoiser preimplemented in Matlab as $\mathcal{D}$. The repeated denoising completely distorts the image. Even more strikingly, the range of the image increases from values in $[0, 1]$ to an interval of approximately $[-185, 218]$ within the first 100 iterations. Clearly, the algorithmic scheme diverges.

A natural condition for the provable convergence of a scheme like [equation \(5.6\)](#) would be a 1-Lipschitz continuous operator $\mathcal{D}$. There has been previous work on computing upper bounds for the best (i. e. lowest provable) Lipschitz constant of a network and using it to enforce a user-defined Lipschitz constant L during training time [[Gou+21](#); [SGL18](#); [Miy+18](#)], but experiments show that enforcing non-expansiveness drastically decreases the denoising performance. The problem of computing even lower bounds for the best Lipschitz constant, in hope of improving those results, was proved to be NP-hard [[VS18](#)] and thus infeasible for deep networks.

Convergence criteria

¹ <https://faces.mpgdl.mpg.de/>

Therefore, the approach described here adapts the idea proposed in [MMC19] to safeguard neural networks by forcing them to predict a descent direction to a given model-based energy, such that it can be used within a line-search algorithm to guarantee convergence. More precisely, at any given estimate $\mathbf{u}$ and model-based energy E the authors use the Euclidean projection onto the half-space

$$\mathcal{C}(\gamma, \nabla E(\mathbf{u})) = \{\mathbf{d} \mid \langle \mathbf{d}, \nabla E(\mathbf{u}) \rangle \geq \gamma \|\nabla E(\mathbf{u})\|_2\}, \quad \gamma > 0 \quad (5.7)$$

after the last layer of their network. Although the resulting algorithm converges to the minimizer of E , experiments showed significantly higher peaks of the PSNR value in early iterations compared to classical gradient descent on E . Intuitively, the descent direction proposed by the network pushes the iteration closer toward the distribution of the training data than a usual gradient descent step would.

While the approach of [MMC19] has to train a separate network for each inverse problem and each type of noise, the method proposed here investigates the combination of the flexible algorithmic schemes in equations (5.5) and (5.6) with the idea of [MMC19] to project onto the half-space of descent directions to safeguard the underlying algorithm.

In the following $\mathcal{D}$ will always refer to a generic, pretrained denoising network, like DnCNN [Zha+17a]. It is assumed that $E(\mathbf{u}) = H_f(A\mathbf{u}) + R(\mathbf{u})$ is a continuous, strictly convex and coercive energy function. As a first step, rewrite the algorithmic schemes equations (5.5) and (5.6) in such a way, that they resemble a gradient descent iteration, i. e.,

$$\mathbf{u}_{t+1} = \mathbf{u}_t - \left(\mathbf{u}_t - \frac{1}{2}\rho(\mathbf{u}_t; 2\tau) - \frac{1}{2}\mathcal{D}(\mathbf{u}_t) \right), \quad (5.8)$$

$$\mathbf{u}_{t+1} = \mathbf{u}_t - (\mathbf{u}_t - \mathcal{D}(\rho(\mathbf{u}_t; \tau))), \quad (5.9)$$

such that $\mathbf{u}_t - \frac{1}{2}\rho(\mathbf{u}_t; 2\tau) - \frac{1}{2}\mathcal{D}(\mathbf{u}_t)$ or $\mathbf{u}_t - \mathcal{D}(\rho(\mathbf{u}_t; \tau))$ can be interpreted as *update directions* of the respective algorithmic schemes. Because the plain iterations can be easily divergent, they are safeguarded by projecting them onto the half space of descent directions $\mathcal{C}(\gamma, \nabla E(\mathbf{u}_t))$, i. e.,

$$\mathbf{d}_t = \text{proj}_{\mathcal{C}(\gamma, \nabla E(\mathbf{u}_t))} (\mathbf{u}_t - \alpha\rho(\mathbf{u}_t; 2\tau) - (1 - \alpha)\mathcal{D}(\mathbf{u}_t)), \quad (\text{conv})$$

$$\mathbf{d}_t = \text{proj}_{\mathcal{C}(\gamma, \nabla E(\mathbf{u}_t))} (\mathbf{u}_t - \mathcal{D}(\rho(\mathbf{u}_t; \tau))). \quad (\text{prox})$$

Note, that the simple averaging of the gradient descent and denoising step in equation (5.5) was replaced by an arbitrary convex combination using a parameter α to more flexibly determine the respective influence of the data term and the denoising term. After computing the above directions $\mathbf{d}_t$, the iterates are updated using

$$\mathbf{u}_{t+1} = \mathbf{u}_t - \eta_t \mathbf{d}_t, \quad (5.10)$$

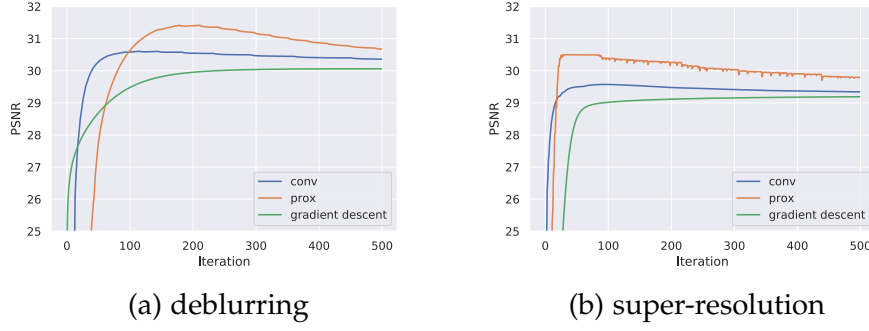


Figure 5.2: Exemplary PSNR values per iteration for the deblurring and super-resolution experiments.

with a step size η_t chosen based on a backtracking line-search mechanism similar to [MMC19]. Under weak additional conditions, the line search guarantees convergence of the proposed scheme to the minimizer of E . Such a minimizer could of course be determined by any classical algorithm, but the aim is for equation (5.10) to yield a better path toward the true minimizer, and thus a *discrepancy principle* for stopping the iteration early is considered. More precisely, the iteration is terminated as soon as

*Discrepancy
principle*

$$H_f(\mathbf{A}\mathbf{u}_t) \leq \beta H_f(\mathbf{A}\hat{\mathbf{u}}), \quad (5.11)$$

with $H_f(\mathbf{A}\hat{\mathbf{u}})$ being an estimate of the data-term dependent noise level of the considered problem, and β being a scaling factor (typically close to 1). This stopping criterion guarantees that the output of the algorithm is close to the physical model represented by the data fidelity term.

5.3 RESULTS

The implementation was tested with the image reconstruction tasks of Gaussian deblurring with standard deviation 1 and $4\times$ single image super-resolution. In both cases, Gaussian noise with standard deviation 0.02 was added to the corrupted image. A PyTorch implementation² of DnCNN [Zha+17a] pretrained on a noise level of 0.1 was used as the denoising network.

The surrogate energy function uses a TV regularization with Huber-norm instead of the L1-norm. The standard data term $H_f(\mathbf{A}\mathbf{u}) = \frac{1}{2}\|\mathbf{A}\mathbf{u} - \mathbf{f}\|_2^2$ was used, with $\mathbf{A}$ either being the blurring or downsampling operator. The best hyperparameter for all methods were found with a grid search. In all experiments, for scheme (conv), $\alpha = 0$ was the best choice for any τ , indicating that the gradient descent step

² <https://github.com/SaoYan/DnCNN-PyTorch>

Method	Camera	House	Pepper	Starfish	Butterfly	Plane	Bird	Lena	Barbara	Boat	Man	Couple
conv, $\beta = 1$	27.47	32.76	29.08	29.45	30.45	27.53	28.45	32.52	25.21	30.04	30.78	29.84
conv, $\beta = 0.9$	28.40	33.32	30.57	29.79	31.21	28.58	29.30	33.49	25.58*	30.79	31.34	30.50
conv, best	28.44	33.36	30.67	29.86	31.23	28.62	29.32	33.76	25.61	30.84	31.41	30.62
prox, $\beta = 1$	28.01	32.17	27.72	29.44	29.96	27.46	29.14	32.73	25.85	30.71	30.78	30.50
prox, $\beta = 0.9$	29.47	34.21	31.05	31.33	32.17	29.12	30.05	34.16	25.64*	31.64	31.87	31.41
prox, best	29.48	34.33	31.40	31.33	32.18	29.16	30.05	34.30	26.88	31.66	31.88	31.42
GD, best	28.19	32.90	30.15	29.55	30.76	28.23	28.95	33.24	25.63	30.64	31.18	30.48

Table 5.1: PSNR for deblurring for varying images and stopping criteria. The algorithm was stopped when $H_f(u_t) < \beta H_f(\hat{u})$. *best* refers to the highest PSNR over 500 iterations and a "*" means that the stopping criterion was not triggered, so the last iteration was used instead.

on the data term does not yield much additional information, assumably because the projection onto $\mathcal{C}(\gamma, \nabla E(u))$, which depends on the gradient of the data term anyway. When using (prox), $\tau = 30$ was empirically found to be the best choice. For the projection onto the half space, $\gamma = 5$ was used in both schemes for deblurring, and $\gamma = 50$ using (conv) and $\gamma = 1$ using (prox) for super-resolution. For fairness, the classical gradient descent baseline was also implemented using backtracking line search.

Figure 5.2 depicts the reconstruction quality of the current iterate compared to ground truth over a span of 500 iterations. The peak signal-to-noise ratio (PSNR) quickly peaks before slowly converging to the fixed point of the surrogate energy, which is consistent with the results of [MMC19]. Notably, the convex combination method peaks earlier, but not as high as the proximal gradient method. Tables 5.1 and 5.2 show results using early stopping using a discrepancy principle. On all test images the (prox) scheme beats gradient descent.

Method	Camera	House	Pepper	Starfish	Butterfly	Plane	Bird	Lena	Barbara	Boat	Man	Couple
conv, $\beta = 1$	21.56	24.10	22.13	22.06	22.20	20.74	20.85	24.88	21.97	25.22	26.35	24.94
conv, $\beta = 0.9$	21.56	24.10	22.13	22.06	22.20	21.27	21.37	28.36	23.40	25.31	26.48	25.02
conv, best	23.00	27.50	23.67	23.60	23.53	22.34	22.50	28.49	23.45	25.32	26.56	25.06
prox, $\beta = 1$	23.43	28.57	24.61	24.41	24.98	22.86	23.27	29.21	23.60	25.87	26.88	25.39
prox, $\beta = 0.9$	23.50	28.78	24.62	24.54	25.09	22.91	23.35	29.38	23.67	25.96	27.05	25.40
prox, best	23.52	28.87	24.72	24.58	25.18	22.93	23.46	29.45	23.72	26.00	27.16	25.53
GD, best	22.21	26.55	22.94	23.01	21.85	21.87	21.40	27.65	23.22	24.94	26.08	24.82

Table 5.2: PSNR for super resolution for varying images and stopping criteria. The algorithm was stopped when $H_f(u_t) < \beta H_f(\hat{u})$. "best" refers to the highest PSNR of all 500 iterations.

5.4 CONCLUSIONS

This chapter combined deep learning and energy minimization methods for solving inverse problems in image reconstruction into a provably convergent algorithmic scheme. Still, the approach is able to

generalize to different problems with a single denoising network and without the need to retrain if the problem changes. Using a discrepancy principle, the method is able to reach better results than the energy minimization baseline in the experiments.

The presented method is limited to smooth energies, and an extension to the non-smooth case is beneficial for certain tasks such as salt and pepper denoising. A direct extension of [MMC19] to non-smooth energies was recently proposed [DMM22], however, this method requires retraining of the network when the energy function changes, like it is the case when the task changes. An extension of this work to denoising priors is an interesting direction for future work.

6

DEPTH ENHANCEMENT WITH GEOMETRIC CONSTRAINTS

This chapter proposes a generic depth map refinement scheme based on techniques from GeoNet, a recent deep learning approach for predicting depth and normals from a single color image, and extends it to be applied to any depth reconstruction task such as super-resolution denoising and deblurring, as long as the task includes a depth output. The approach utilizes a tight coupling of the inherent geometric relationship between depth and normal maps to guide a neural network. In contrast to GeoNet, the method does not utilize the original input information to the backbone reconstruction task, which leads to a generic application of the network structure. The approach first learns to output a high-quality normal map from the depth image generated by the backbone method, and then refines this normal map jointly with the initial depth image. This is motivated by the fact that it is difficult for neural networks to learn a direct mapping between depth and normal maps without explicit geometric constraints. Experiments show the efficiency of the method on the exemplary depth image reconstruction tasks of denoising, super-resolution, and removal of motion blur. The method presented in this chapter was published in [SK22].

6.1 INTRODUCTION

High-quality depth maps are required in a wide variety of computer vision and graphics tasks, such as RGB-D scene reconstruction [Kel+13; Dai+17], augmented reality [LPW11; HK18; Du+20] and autonomous driving [You+20; AR20]. Compared to standard RGB sensors, depth sensors often produce noisy images, which makes depth reconstruction tasks especially challenging, since each task also needs to account for the different depth uncertainties or deficiencies specific to the task. Some classes of sensors suffer from artifacts that are uncommon in typical color sensors. For example, artifacts from motion relative to the camera are a particular problem for ToF cameras, because they capture multiple phase images in sequence (see Section 2.1.2.6). Solutions for these problems require specialized algorithms, such as those outlined in [Ste+19a].

Although approaches that are well known in the realm of color image enhancement, such as energy minimization techniques (Section 2.3.1) or deep learning (Section 2.2), can often be translated one-to-one to depth enhancement tasks, usually by just interpreting

the depths as grayscale values, this fails to incorporate the inherent geometric structure depth maps represent. While research on depth-only enhancement exists [RRB16; LGW19], a majority of recent work has focused on some form of intensity or RGB-guided depth enhancement, e. g., for super-resolution [HLT16; KPH21; TCZ21; Guo+18], denoising [Ste+19b; Yan+18] or motion blur removal [Tou+16; Li+20]. While this greatly improves the quality of the resultant depth images, these additional RGB sensors are not always available. Moreover, none of the above examples explicitly incorporates surface normal information, which is tightly geometrically linked to depth information. However, in the area of monocular depth estimation, i. e., depth estimation from a single RGB-image, there have recently been works that not only produce normal maps as an additional output but also successfully use them to enhance the quality of the final depth map [Qi+18; Qi+20; EF15; Xu+18; Wan+16]. Most notably, Qi et al. [Qi+18; Qi+20] introduce the GeoNet/GeoNet++ network architecture to estimate a depth and a normal map from a single RGB image. Their approach toggles between a depth-to-normal module, which utilizes a least squares approach, and a normal-to-depth module, which estimates depth based on kernel regression in order to enforce geometric consistency between the two domains. Their approach can be seen as a weak coupling between normals and depth, as the two stages operate independently. Still, GeoNet++ outperforms standard CNN approaches that learn a direct mapping between depth and surface normals, both in terms of accuracy and normal-depth consistency. In an ablation experiment, the authors show that CNNs generally have problems to learn a direct mapping between depth and surface normals in a consistent way [Qi+20]. Since it is already hard to learn this mapping in a supervised setting with normals as output, it can be hypothesized, that neural networks also have difficulties including surface normal information in their latent representations without explicit geometric constraints.

This chapter develops a generic depth refinement scheme, which takes surface normals into account but makes no assumptions about the specific task that is to be solved, except that the output is a depth map. Based on the GeoNet/GeoNet++ concept, the approach computes high-quality normal maps in an intermediate step, which are then used to refine an initial depth estimate provided by the backbone method. Contrary to GeoNet, the method does not utilize the original input to the backbone method, making the approach generic to many existing reconstruction methods. Moreover, the approach uses a tighter coupling between the depth and the normal domain by linking both stages using skip connections, making full normal and depth information available at depth-to-normal and normal-to-depth stages.

Contributions

Experiments show that this approach improves the quality compared to existing methods in a variety of different tasks, namely depth-only super-resolution, RGB-guided super-resolution, additive Gaussian noise removal and deblurring.

6.2 RELATED WORK

This section will give a brief overview of research in different areas of depth enhancement. The methods can be roughly split into classical variational methods and learning-based methods.

Specialized variational and classical non-learning-based approaches for depth enhancement generally aim to improve depth data with additional sensor data, like color images. Huhle et al. [Huh+08] use a non-local means (NLM) approach to remove outliers from depth data by computing an additional color-based weight in their NLM formulation. Ferstl et al. [Fer+13] use a variational approach to compute higher-resolution depth images with the help of already high resolution intensity images. Some approaches specialize in specific sensor types: Shen and Cheung [SC13] introduce a probabilistic model using a Markov random field for denoising and completing depth maps from structured light sensors. Another work on structured light sensors was presented by Fu et al. [Fu+12], who specifically target the spatiotemporal denoising of the Microsoft Kinect depth camera.

Classical methods

In recent years, like in any other field of computer vision and graphics, there has been substantial amounts of deep learning research for depth reconstruction. The work from Tourani et al. [Tou+16] deals with the removal of motion artifacts from rolling shutters, which are common in structured sensors such as the Kinect. Li et al. [Li+20] use a two-branched CNN to simultaneously remove motion blur from a color and a depth image. The problem of depth-only super-resolution, i. e., without additional color data, was tackled by Li et al. [LGW19], who extend ideas from deep Laplacian pyramid networks [Lai+17], which were originally proposed for RGB super-resolution, to depth. They put their work into the context of 3D reconstruction, which they show can greatly benefit from higher-resolution depth images. Research in the area of color-guided depth super-resolution is more widespread. Zhao et al. [Zha+19] jointly upscale depth and color images by using a generative adversarial network (GAN). Another deep learning approach was proposed by [KPH21] in the shape of deformable kernel networks for joint image filtering. Apart from guided depth image super-resolution their approach can also be applied to saliency map upsampling, cross modality image restoration and texture removal. Recently, Tang et al. [TCZ21], inspired by progress in neural implicit representations, introduced joint implicit image functions (JIIF) and interpreted the problem of guided depth super-

Deep learning

resolution as a neural implicit interpolation task. Another recent deep learning approach is by Zhong et al. [Zho+21] who used an attention-based network design to fuse the most important features from depth and color images, and then use those features to guide an upscaling network.

Even though the experiments in this chapter do not include monocular depth prediction tasks, works from this field that use explicit surface normal information are also related to the presented approach. Apart from GeoNet by Qi et al. [Qi+18; Qi+20], which the method proposed in this chapter directly extends and will be discussed in more detail in the upcoming sections, some other research in that direction will be listed here. Eigen and Fergus [EF15] tackle the task of predicting depth, normals, and semantic segmentation from RGB images in a single deep neural network. Xu et al. [Xu+18] first predict the initial depth, surface normal, semantic segmentation, and contour maps, and then fuse them into a final high quality depth image. However, in both of these works, there is no enforcement of consistency between the predicted normal and depth images. A more tightly coupled approach was proposed by Wang et al. [Wan+16], who introduced an orthogonal compatibility constraint between normals and surface points that lie in a common planar region. However, their computations are very costly and the method might fail in non-planar regions of the scene.

Monocular depth estimation

6.3 METHOD

This section introduces the generalized depth enhancement framework for arbitrary depth reconstruction tasks. It starts with a brief review of the main ideas from GeoNet [Qi+18; Qi+20] in Section 6.3.1. In Section 6.3.2, these ideas are extended to the general depth enhancement network. Finally, the loss functions used for training are discussed in Section 6.3.3, and implementation details are discussed in Section 6.3.4.

6.3.1 GeoNet

Originally, GeoNet is a method for estimating a normal and a depth map from a single RGB image. In the following explanations of GeoNet, it is assumed, that initial normal and depth estimates, by whichever means, e. g., another CNN, have already been computed. The initial normal at pixel i is denoted as $\mathbf{n}_i^{\text{initial}}$ and the initial depth at pixel i as z_i^{initial} . Further following the notation of Qi et al. [Qi+18; Qi+20], pixel coordinates are denoted as (u_i, v_i) and corresponding 3D

coordinates as (x_i, y_i, z_i) . The mapping between the two is determined by the pinhole camera equation (Section 2.1.1.2)

$$x_i = (u_i - c_x)z_i / f_x \quad y_i = (v_i - c_y)z_i / f_y, \quad (6.1)$$

where f and c are intrinsic camera parameters.

The main idea of GeoNet is now to refine the initial normal map by using the geometric constraints given by the depth map, and vice versa. This is motivated by the fact that both representations have an inherent geometric relationship with each other that is difficult to learn directly through a network. In the following, both paths, i.e., depth refinement using normals and normal refinement using depth, will be discussed separately.

6.3.1.1 Normal refinement

To refine the initial normal map $\mathbf{n}^{\text{initial}}$, first an additional normal map that is consistent with the initial depth map is computed. To avoid confusion, this auxiliary normal map will be denoted as $\mathbf{n}^{\text{depth}}$. By using the assumption, that surface points in a local neighborhood approximately lie on the same plane, $\mathbf{n}^{\text{depth}}$ can be computed from z^{initial} by first projecting the local neighborhood back into 3D using equation (6.1) and then computing the normals using least squares. The neighborhood of size β around i is defined as

$$N_i = \{(x_i, y_i, z_i) \mid |u_i - u_j| < \beta, |v_i - v_j| < \beta, |z_i - z_j| < \gamma z_i\}, \quad (6.2)$$

where γ is a parameter to filter out depths which deviate too much from the center depth. Writing the points of this neighborhood into a matrix

$$\mathbf{A} = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_K & y_K & z_K \end{pmatrix} \in \mathbb{R}^{K \times 3}, \quad (6.3)$$

enables the calculation of the normals as the least squares solution

$$\mathbf{n}_i^{\text{depth}} = \frac{(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{1}}{\|(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{1}\|}. \quad (6.4)$$

Here, $\mathbf{1}$ is the K -dimensional constant vector only containing 1s. Since this normal is prone to noise, it is further refined by a residual network (see Section 2.2.2.5), that also takes $\mathbf{n}^{\text{initial}}$ as input. In [Qi+20], the network is defined as

$$\mathbf{n}^{\text{final}} = \mathcal{G}_2((\mathcal{G}_1(\mathbf{n}^{\text{depth}}) + \mathbf{n}^{\text{depth}}) \circ \mathbf{n}^{\text{initial}}), \quad (6.5)$$

where $\mathcal{G}_1$ and $\mathcal{G}_2$ are CNNs and $\circ$ means concatenation along the channel dimension. The residual network $\mathcal{G}_1$ is implemented as a residual network with an additive skip connection between the output and the input normal map (see Section 2.2.2.5). The output of the final network $\mathbf{n}^{\text{final}}$ is the refined normal map.

6.3.1.2 Depth refinement

Analogous to the previous section, the first step is to compute a depth map z^{normal} that is consistent with the initial normal map $\mathbf{n}^{\text{initial}}$. The assumption is the same: points in a close neighborhood lie on the same plane. The neighborhood around pixel i is defined as

$$M_i = \{(x_j, y_j, z_j) \mid |u_i - u_j| < \beta, |v_i - v_j| < \beta, \langle \mathbf{n}_i, \mathbf{n}_j \rangle > \alpha\}. \quad (6.6)$$

Instead of filtering out large depth deviations, normals with a large angular difference to the center normal are filtered out.

Given only the center depth, the depth for each point in the neighborhood can now be estimated as

$$z'_{ji} = \frac{n_{jx}x_j + n_{jy}y_j + n_{jz}z_j}{(u_i - c_x)n_{jx}/f_x + (v_i - c_y)n_{jy}/f_y + n_{jz}}. \quad (6.7)$$

These depth estimates are then aggregated by a weighting term dependent on the angular difference of their normal to the center normal by kernel regression

$$z_i^{\text{depth}} = \frac{\sum_j \langle \mathbf{n}_j, \mathbf{n}_i \rangle z'_{ji}}{\sum_j \langle \mathbf{n}_j, \mathbf{n}_i \rangle}. \quad (6.8)$$

Again, these rough estimates are further refined with a [CNN](#)

$$z^{\text{final}} = \mathcal{G}_3(z^{\text{depth}} \circ z^{\text{initial}}). \quad (6.9)$$

Note that all operations above, particularly computing least squares solutions and kernel regressions, are differentiable, which means all networks, including the upstream RGB-to-depth network, can be trained end-to-end.

6.3.2 General depth enhancement network

This section extends the ideas from GeoNet [Qi+20] from its monocular depth estimation task to arbitrary depth-to-depth refinement tasks, under the assumption, that there is some generic algorithm $\mathcal{D}$ (such as a neural network) that maps an input depth x (e.g., a low-resolution depth map) to an initial depth estimate of its specific task (such as super-resolution), i.e.,

$$z^{\text{initial}} = \mathcal{D}(x). \quad (6.10)$$

In the following, $\mathcal{D}$ is referred to as the *backbone (network)*, but note that, despite the experiments in this chapter only including neural networks as choices for $\mathcal{D}$, no assumptions on the structure of differentiability are made, i.e., it could also be a classical image reconstruction method such as non-local means or energy minimization.

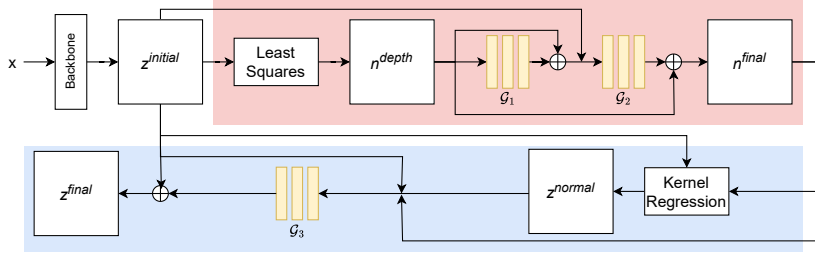


Figure 6.1: The proposed network architecture. Red: normal refinement module; blue: depth-refinement module; yellow: CNNs with learnable parameters

Unlike GeoNet, which also requires an additional backbone for initial normal computation, the proposed method only requires a backbone that maps x to an initial depth estimate. Moreover, the approach does not utilize the original input data x to the backbone, making it independent of the underlying refinement task. Instead of having the two independent depth and normal refinement branches, a single sequential refinement scheme is employed, in which at first a high-quality normal map from the initial depth estimate is computed and then used to refine the depth map again, similar to the iterative toggling scheme in [Qi+20].

Equation (6.4) is used to calculate a rough normal estimate n^{depth} . Unlike in equation (6.5), the initial depth is also concatenated to the output of the refinement network and additional skip connections are added. Compared to GeoNet, these residual connections enable a tighter coupling of depth and normal information in both stages

$$n^{\text{final}} = \mathcal{G}_2((\mathcal{G}_1(n^{\text{depth}}) + n^{\text{depth}}) \circ z^{\text{initial}}) + n^{\text{depth}}. \quad (6.11)$$

These normals are then used to refine the depth map again. The idea here is that first guiding the network to learn accurate normals might help it to find geometric structure that it would otherwise have missed.

The intermediate depth estimation z^{normal} is calculated with equations (6.7) and (6.8), and further refined into the final result by applying another CNN. Again, additional skip connections are added:

$$z^{\text{final}} = \mathcal{G}_3(z^{\text{normal}} \circ n^{\text{final}} \circ z^{\text{initial}}) + z^{\text{initial}}. \quad (6.12)$$

The overall architecture of the scheme is visualized in Figure 6.1. Many parts of the architecture are fixed functions without learnable weight (like kernel regression and least squares), which makes the training converge quickly. The concrete implementation of the CNNs, like the number of layers and kernel sizes, will be discussed in Section 6.3.4.

6.3.3 Loss functions

Analogously to GeoNet, every operation from the initial depth estimate z^{initial} to the refined estimate z^{final} is differentiable. This means

Network	Parameters	Time (ms)
DnCNN	556k	26
ADNet	519k	28
ResNet	556k	26
DLapSRN	435k	10
DKN	1.4M	126
+Proposed	+235k	+34

Table 6.1: Learnable parameters and timings of the networks used in the experiments. Timings were measured using a Nvidia Tesla V100 GPU and AMD EPYC 7452 CPU on a single 640×480 image (output size) from the NYU v2 dataset [Sil+12].

all networks (including the backbone, if it is also a neural network) can potentially be trained in an end-to-end fashion. However, in the experiments in this chapter, to showcase the generality of the approach, the backbones are pretrained and their weights are frozen before training the remaining network. The loss functions are computed on the intermediate results and all individual loss terms are summed up to the total loss function $\mathcal{L} = \mathcal{L}_{\text{normal}} + \mathcal{L}_{\text{depth}}$. More specifically, the normal loss term is defined like in [Qi+20]:

$$\mathcal{L}_{\text{normal}} = \sum_i \|n_i^{\text{final}} - n_i^{\text{gt}}\|_2^2 + \lambda \sum_i \|n_i^{\text{depth}} - n_i^{\text{gt}}\|_2^2. \quad (6.13)$$

The depth loss has a few modifications compared to Qi et al. [Qi+20]. The loss is not applied to the direct output of the backbone, since its weights are frozen. Instead, the loss is also computed on z^{normal} . Even though there is only the kernel regression step with no learnable parameters between the computations of n^{final} and z^{normal} , experiments have shown that it is still beneficial to have this additional loss term to pass gradients to the upstream networks inside the depth-to-normal module. A smoothed L1-loss is used [Cha+94]:

$$\mathcal{L}_{\text{depth}} = \sum_i \sqrt{(z_i^{\text{final}} - z_i^{\text{gt}})^2 + \epsilon} + \eta \sum_i \sqrt{(z_i^{\text{normal}} - z_i^{\text{gt}})^2 + \epsilon} \quad (6.14)$$

The same smoothed L1-loss used to pretrain the backbone networks.

Note, that ground truth normals are only required during training, but not during inference. This allows the use of the network on arbitrary backbones as long as they output depth images.

6.3.4 Implementation details

The same network architecture is used on top of each backbone. Each CNN in the depth refinement scheme (see Figure 6.1) consists of just four convolutional layers with kernel size 3 and 64 channels, which

results in 235k additional learnable parameters. Table 6.1 shows the parameters and runtime overhead of the depth refinement scheme compared to different backbone networks. ReLU activations are used in every CNN and the chosen hyperparameters are $\eta = 0.5$ and $\lambda = 10^{-3}$ for the loss weighting, and $\epsilon = 10^{-6}$ for the L1 loss smoothing factor. The neighborhood size in equations (6.2) and (6.6) is set to 9×9 and the filtering parameters are chosen as $\gamma = 0.05$ and $\alpha = 0.95$. Images are center-cropped to a size of 256×256 , randomly flipped along the vertical axis for data augmentation and the networks are trained with a batch size of 16. The weights of the networks are frozen in all experiments. In general, only 2-3 additional epochs are needed for the training of the depth enhancement model to converge.

6.4 EVALUATION

6.4.1 Training and tasks

The effectiveness of the depth enhancement method is demonstrated on a variety of classic image reconstruction tasks. To show the general nature of the approach, it is added and compared to several different backbone networks. All backbones were trained from scratch using code provided by their respective authors and using the training data provided by Qi et al. [Qi+18]. The dataset is based on the NYU v2 dataset Silberman et al. [Sil+12] and contains 30,816 frames with real-world depth and color images taken with a Microsoft Kinect camera, as well as high-quality normal maps that were used as ground truth. The input to the networks was simulated from the ground truth images with the forward operators of the respective tasks (see Section 2.3) and will be further detailed in the following sections.

6.4.1.1 Denoising

The first task used for comparison is the removal of additive Gaussian noise with known variance. The chosen backbone networks are DnCNN [Zha+17a] and ADNet [Tia+20]. The randomly sampled Gaussian noise with the high standard deviation of 0.5 m is added to the ground truth depth images and the networks are trained with default parameters.

6.4.1.2 Deblurring

The ground truth depth was convolved with a 25×25 blur kernel that contained zeros everywhere except on the main diagonal, where it was set to a constant $1/25$. This roughly simulates motion blur of a far-away scene with a camera rotating diagonally between the top

Task	Method	RMSE ↓	MAE ↓	SSIM ↑
Denoising	DnCNN	4.07	2.84	0.9663
	DnCNN + Ours	3.81	2.57	0.9757
	ADNet	3.64	2.47	0.9730
	ADNet + Ours	3.55	2.34	0.9743
Deblurring	ResNet	3.14	2.14	0.9897
	ResNet + Ours	2.97	2.00	0.9896
Super-resolution (Depth only)	Bilinear	3.63	1.09	0.9821
	Bilinear + Ours	3.07	0.93	0.9849
	DLapSRN	2.85	0.88	0.9863
	DLapSRN + Ours	2.26	0.71	0.9889
Super-resolution (RGB guided)	DKN	1.68	0.61	0.9931
	DKN + Ours	1.59	0.59	0.9936

Table 6.2: Quantitative comparison of depth map reconstruction for different tasks on the NYU v2 dataset [Sil+12]. Values are given in centimeters (RMSE and MAE) and averaged over all test set images.

right and bottom left image corners. The used backbone was a 17-layer ResNet [He+16] which has a similar architecture to DnCNN [Zha+17a].

6.4.1.3 Super-resolution

The experiments include methods both from depth-only super-resolution as well as color-guided super-resolution. For the former, DLapSRN [LGW19] was used, which is itself based on Laplacian pyramid networks [Lai+17]. The backbone for color-guided super-resolution is the recent deformable kernel network (DKN) [KPH21]. Bilinear filtering was employed to sub-sample ground truth images to a factor of 1/4. Again, all networks were trained with default parameters until convergence. In order to evaluate how the network deals with input of lower quality, the depth enhancement network was also trained with a simple bilinear interpolation backend.

6.4.2 Results

The different methods were evaluated on a separate 654 image subset of the common NYU v2 benchmark dataset Silberman et al. [Sil+12], which is often used to evaluate super-resolution tasks [KPH21; RRB16; TCZ21]. Since there are no such commonly used benchmark datasets for depth map denoising and deblurring, the same dataset was used for the evaluation of all tasks. Quantitative results can be found in

Task	Method	RMSE ↓	MAE ↓	SSIM ↑
Denoising	DnCNN	6.21	4.49	0.8932
	DnCNN + Ours	5.55	3.82	0.9427
	ADNet	5.32	3.70	0.9200
	ADNet + Ours	5.09	3.44	0.9383
Deblurring	ResNet	3.06	1.74	0.9574
	ResNet + Ours	2.96	1.67	0.9581
Super-resolution (Depth only)	Bilinear	2.54	1.00	0.9629
	Bilinear + Ours	2.35	0.93	0.9663
	DLapSRN	2.03	0.85	0.9696
	DLapSRN + Ours	1.69	0.74	0.9749
Super-resolution (RGB guided)	DKN	1.23	0.61	0.9805
	DKN + Ours	1.09	0.60	0.9806

Table 6.3: Quantitative comparison of depth map reconstruction for different tasks on the Middlebury dataset Hirschmuller and Scharstein [HS07]. Values are given in pixel disparity provided by the dataset.

Table 6.2. The add-on network consistently improved the results of all backbone networks in terms of root mean square error (RMSE) and MAE. Since the exact same backbone was used in the comparison, this improvement has to be a result of the depth refinement scheme. The scheme improves RMSE in a range from 3% for ADNet up to 20% for DLapSRN, and improves MAE in a range of 6% to 20%. This discrepancy could be explained with the quite challenging noise level, with a standard deviation of 0.5 m, in the denoising experiments. Since the outputs of the backbones still include many defects, the initial normal computation could also output low-quality normals that are not as helpful to the depth refinement network. Note, that the add-on-like approach with a residual connection between the backbone output and the final result helps the refinement scheme to be at least of the same quality as the backbone output, since in the worst case the network could just learn to output the initial depth image. In terms of structural similarity index measure (SSIM) [Wan+04], the depth enhancement scheme performs slightly worse than the baseline in the deblurring experiment, but in general the margins are lower, with the exception of the DnCNN experiments. Note, that the networks were not explicitly optimized for perceptual quality.

To show that the depth refinement is able to generalize to new datasets, it was also evaluated on 30 images of the Middlebury stereo dataset [HS07] without fine tuning the networks. The Middlebury dataset contains pixel disparity images which other authors [RRB16;

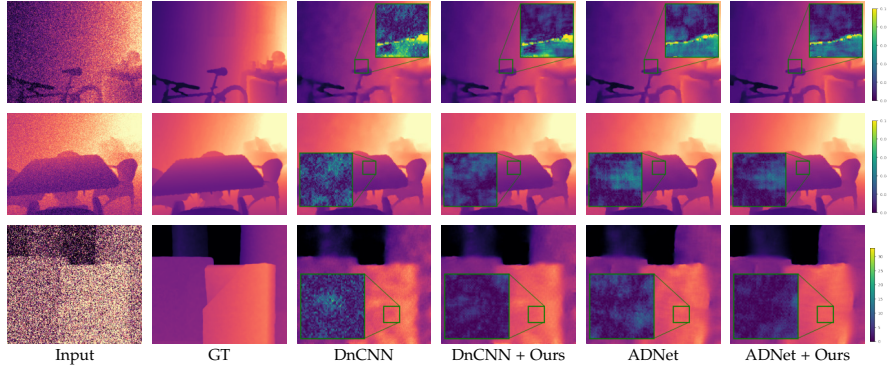


Figure 6.2: Denoising results. Insets show absolute difference to ground truth. The first two rows are examples from the NYU v2 [Sil+12] test set and the third row from the Middlebury dataset [HSo7]. Note that while, like other authors, inputs and results of the Middlebury example are shown as pixel disparity values, the noise is added to the converted depth maps.

[LGW19; KPH21; TCZ21] directly interpret as depth values before feeding them into their methods. Since normal map computation requires reprojecting the depth values back into 3D space, the disparity values need to be converted into real depth before inputting them into the backbones. To make the results comparable to other methods, the final depth maps were converted back into disparity values before evaluation metric computation. The results in Table 6.3 show that the depth refinement add-on again consistently outperforms the baseline methods. The Middlebury dataset uses stereo images, as opposed to the NYU v2 training set which uses structured light [Sil+12]. This is evidence, that all tested methods can generalize to different types of sensors.

Figures 6.2 to 6.5 show qualitative results, including pixel-wise absolute difference to the ground truth depth inside the insets. The areas of the highest improvement differ between tasks. The depth enhancement module improved the denoising backbones mostly in planar regions (Figure 6.2). This might be due to the windowed least squares normal computation inside the depth-to-normal module that acts like a low-pass filter. Nevertheless, sharp edges were still preserved by the network. In contrast, the deblurring (Figure 6.3) and super-resolution (Figures 6.4 and 6.5) already output high-quality planar regions and the improvements of the depth enhancement module are mostly located at the edges. The differences in results for color-guided super-resolution in Figure 6.5 are more subtle. DKN can already achieve very sharp edges by utilizing the color information, while the depth enhancement seems to mostly improve some outliers at those edges.

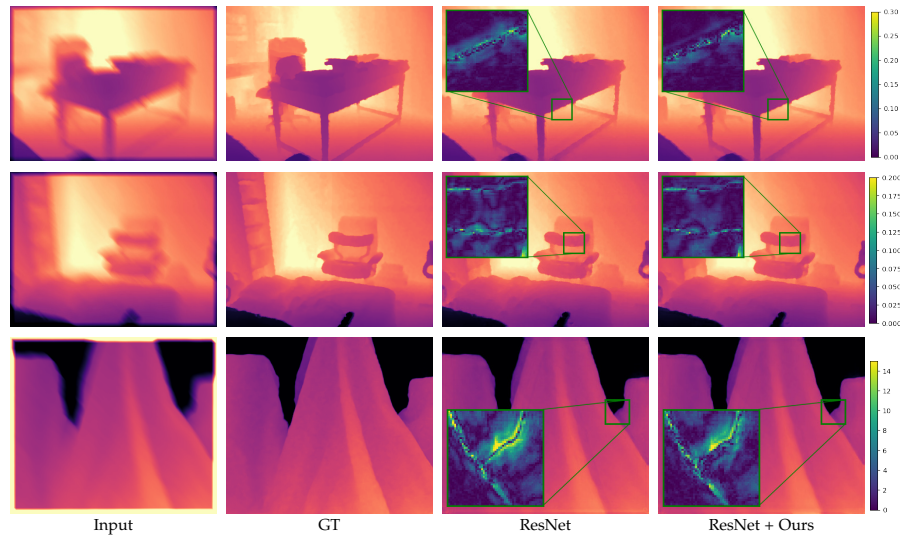


Figure 6.3: Deblurring results. Insets show absolute difference to ground truth. The first two rows are examples from the NYU v2 [Sil+12] test set and the third row from the Middlebury dataset [HS07].

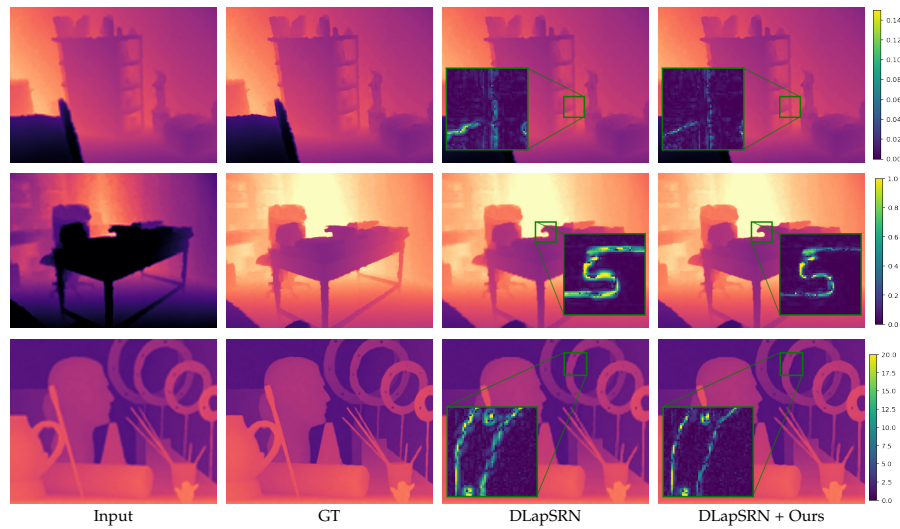


Figure 6.4: Depth-only super-resolution results. The input is upscaled for visualization purposes. Insets show absolute difference to ground truth. The first two rows are examples from the NYU v2 [Sil+12] test set and the third row from the Middlebury dataset [HS07].

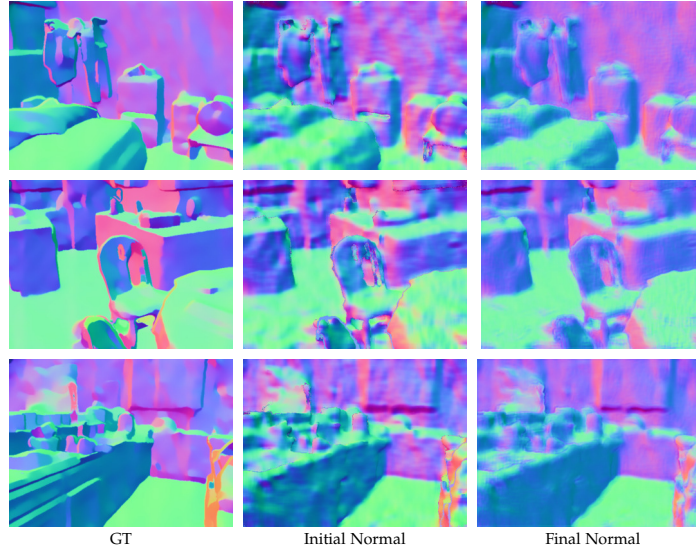


Figure 6.6: Visualization of normal map quality on three scenes of the NYU v2 dataset using the ADNet backbone.

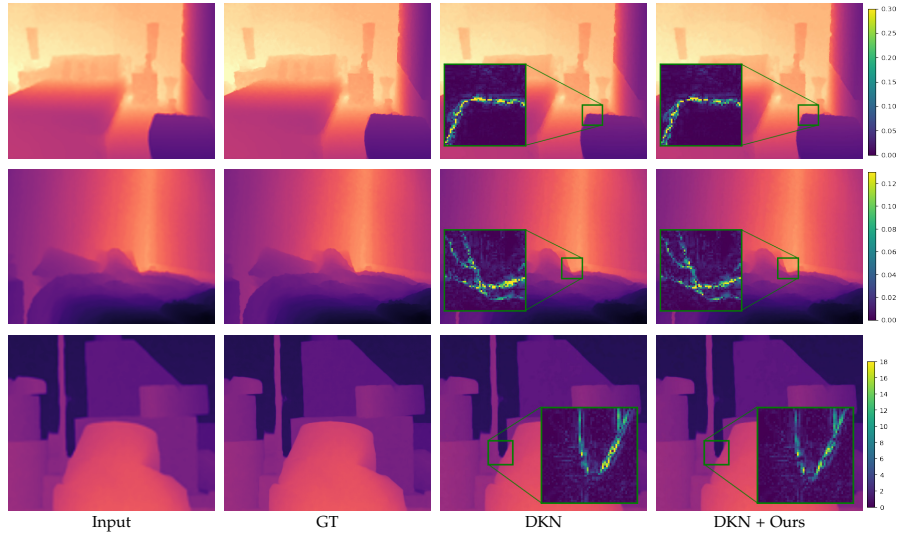


Figure 6.5: RGB-guided super-resolution results. The input is upscaled for visualization purposes. Insets show absolute difference to ground truth. The first two rows are examples from the NYU v2 [Sil+12] test set and the third row from the Middlebury dataset [HS07].

To verify the normal refinement module, exemplary normal map visualizations from the denoising results using the ADNet backbone are shown in Figure 6.6. As suggested above, the normal computation for the denoising task is more challenging than for the other tasks. The final normal maps are significantly less noisy than the normals directly computed with least squares. Especially in planar regions the normals show high levels of noise. Note, that the focus of the depth enhancement module is more on depth map quality while normal

maps were treated more as auxiliary data, and hyperparameters were tuned accordingly.

6.5 DISCUSSION AND CONCLUSIONS

This chapter introduced a generic depth enhancement framework for a potentially wide variety of depth reconstruction tasks. The method is able to improve on several prior learning-based methods by adding just a few additional learnable parameters. The approach has the nice side effect that it also calculates higher-quality normal maps that can be utilized in some tasks, e. g., 3D reconstruction.

There are multiple possible directions for future work. The frozen weights in all of the pretrained backbone networks during training of the depth enhancement module might lead to lower performance compared to training them in tandem. On the other hand, since the output of the different backbones is already similar, the depth enhancement module might be rather easily used for transfer learning. Another possible direction is to apply the depth enhancement iteratively, similar to GeoNet++ [Qi+20]. In essence, the enhanced depth map is used to compute a higher-quality normal map again, which in turn can be used to improve the depth again, and so on.

Future work

7

REAL-TIME RENDERING OF NOISY POINT CLOUDS

This chapter develops a method to handle the real-time visualization of imperfectly measured point clouds. Typical point-based rendering algorithms cannot directly handle large outliers or high amounts of noise without either a costly pre-processing of the point cloud or using multiple render passes. Instead, this chapter proposes an A-buffer-based approach, which directly renders unprocessed point clouds and filters out outliers in real-time, without introducing additional render passes. The core concept of the approach uses bins along each viewing ray in which intermediate information is accumulated. To improve storage efficiency, the binned A-buffer approach is extended with a per-ray bin hashing. The method significantly improves visual quality of noisy point clouds with varying noise levels and large outliers, while only introducing little performance and memory overhead compared to traditional point rendering methods. The approach presented here was published in [SK24].

7.1 INTRODUCTION

Using points as a rendering primitive [LW85] has gained much popularity over the last decades. In recent years, powerful laser scanners and consumer-grade depth measuring devices like ToF cameras such as the Microsoft Kinect make it easier than ever to capture dense point clouds of arbitrary objects and scenes. The unstructured nature of point clouds, without the need to store and keep track of topology information, makes them an attractive choice for applications such as 3D reconstruction [Kel+13; LWK15; Lef+17; Whe+15], robotics [PCS+15] and the visualization of terrestrial data [Don+20; Bie+07].

However, besides lacking topological information, captured point clouds exhibit challenges in terms of non-even sampling density, locally varying point size, as well as sensor noise and outliers. The latter mainly occur as flying pixels from ToF scans (see Section 2.1.2.6) at depth discontinuities, i. e., edges between foreground and background objects. Common point cloud rendering algorithms are not robust to these outliers. Opaque splatting (Section 2.4.2) renders each point as an opaque disk with spatial extend, possibly aligned along a plane defined by a normal that is stored together with each point [RL00; Kel+13]. To make the image hole-free, these splats need to overlap, which necessarily leads to artifacts in high-curvature regions, since only the splat closest to the camera contributes to each pixel.

More sophisticated approaches use [EWA](#) splatting of point attributes (like depth and color) based on the distance of fragments from the splat center, commonly implemented on GPUs as a 3-rendering-pass approach in rasterization pipelines [[Bot+05](#); [BSK04](#)]:

1. Compute the splats closest to the camera for each pixel
2. Accumulate point attributes within a depth range ϵ (surface thickness) along each viewing ray
3. Normalize the accumulated values using per-pixel operations

See [Section 2.4.3](#) for more details on [EWA](#) and its hardware implementation. Although the traditional 3-pass approach works well for homogeneously sampled scenes without large noise and outliers, it cannot handle these kinds of real-world data without either discarding the pixel completely or doing costly additional full rendering passes [[DB07](#)].

The main restriction of using EWA with noisy point clouds is that only points in the vicinity of the closest splat to the camera are taken into account in the subsequent accumulation pass. If the closest splat is an outlier, all fragments belonging to the real surface intersection behind the outlier are not accumulated. However, the question whether a splat is an outlier or not can only be answered after all fragments have been processed, since the order of processed fragments is entirely random in hardware-accelerated rendering pipelines. As the reconstruction of the real surface for real-world point clouds is mainly a statistical problem, solving this issue efficiently requires information of all fragments to be retained, not just the fragments closest to the camera.

This chapter proposes an enhanced 3-pass rendering approach for real-world point clouds that efficiently handles high and varying levels of noise in combination with strong outliers, without requiring any preprocessing. The method borrows from multifragment rendering [[VVP20](#)], which has conceptual similarity to rendering noisy point clouds with outliers, in that a (partial) list of all fragments is retained for each pixel (see [Section 7.2](#)). However, due to memory and runtime requirements, multifragment rendering concepts can not be directly applied to point clouds, which usually have large amounts of pixel overdraw. For this reason, this chapter introduces the *hashed, binned A-buffer* concept, comprising the following main contributions to close this gap:

Contributions

- Direct rendering of *raw, noisy point clouds with varying noise levels*, without the need of preprocessing.
- A binning technique for accumulating weighted contributions of all surface splats along a viewing ray in a *single render pass* without explicit storage and sorting of splat-fragments.

- An efficient implementation of binned A-buffers comprising
 - *per-pixel bin hashing* to improve memory efficiency, and
 - $\mathcal{O}(1)$ *splat insertion and surface retrieval complexity* by tracking the closest valid surface during the accumulation pass without an additional post-processing step.

The overall background motivation is instant rendering of point clouds acquired with hand-held depth cameras, which results in point clouds with varying point density and distance-related noise levels. Therefore, the method is evaluated qualitatively and quantitatively using simulated RGB-D range images.

7.2 RELATED WORK

This section discusses prior work from point-based rendering and multifragment rendering, and outlines the differences to the technique developed in this chapter.

7.2.1 Point-based rendering

The usage of points as rendering primitives was first proposed by Levoy and Whitted [LW85]. Since purely point-based representations require a very dense sampling for hole-free visualization, the idea was extended by Zwicker et al. [Zwi+01] by using object aligned elliptical resampling kernels which are accumulated in screen space. This EWA approach was later mapped to modern programmable graphics hardware by Botsch et al. [Bot+05; BSKo4] using the 3-pass approach described in Section 2.4.3. Although splats are elliptical, sharp features can be rendered by clipping them with precomputed clipping lines [BSKo4; Pau+03], which, however, require preprocessing of the point clouds.

Closely related to the method developed in this chapter is the work by Diankov and Bajcsy [DBo7], which directly renders a noisy point cloud with an iterative surface splatting scheme. The approach iterates over the usual 3-pass scheme 3-5 times, thereby decreasing rendering performance by the same factor.

Another method was proposed by Uchida et al. [Uch+20] in the context of rendering *transparent* surfaces with noisy points and possibly many outliers. However, the method requires costly preprocessing for noise removal and point sorting, and the resulting images cannot contain opaque surfaces due to inherent design restrictions of the method. Similarly, Seemann et al. [See+18] propose to segment a point cloud into *solid* and *soft* points by thresholding a scalar field, like measurement uncertainty. By only rendering the solid points, this method can also be used for outlier removal, but the scalar field computation

again requires a preprocessing step, resulting in an offline outlier removal scheme.

An orthogonal research area is the out-of-core handling and rendering of very large point clouds, which is often accelerated with structures like k -d trees [Gos+13]. Pintus et al. [PGA11] propose a non-splatting out-of-core direct rendering approach for massive raw point clouds using screen-space operators, which does not require special point attributes like normals and radii. They define a point's visibility via the maximal aperture cone in the direction of the camera that does not contain any other point. This approach cannot handle a massive amount of outliers like the method proposed in this chapter.

Recent works have incorporated deep learning approaches into point cloud rendering. Dai et al. [Dai+20] aggregate learnable per-point feature vectors into multiple planes along the camera frustum and subsequently use those feature planes as input to a convolutional neural network trained for rendering. While the results are impressive, the per-point features have to be retrained for each scene, making the method unsuitable for direct point cloud rendering. The Z2P neural rendering approach [Met+22] can handle noisy, raw point clouds of small size only, e. g., it requires 3 seconds for just 5k points. For an additional overview on recent advances in photorealistic rendering of non-noisy point clouds see the survey by Kivi et al. [Kiv+22].

In contrast to prior work, the proposed hashed, binned A-buffer approach avoids additional render passes (as needed by, e. g., [DB07; BSK04]), does not require any preprocessing to remove outliers from raw point clouds, such as [See+18; Uch+20].

7.2.2 Multifragment rendering

Rendering noisy point clouds with outliers requires retaining information from all surfaces, which is a problem closely related to the area of multifragment rendering, which mainly has applications in order-independent transparency, shadows and global illumination. In multifragment rendering, the final pixel value is generated using a (partial) *list* of all fragments per pixel ray and a post-processing reduction step to combine the list of fragments. This approach allows for more or less arbitrarily complex blending operations, making them potentially more suitable for real-time graphics. For a more complete overview of multifragment methods, see the recent survey by Vasilakis et al. [VVP20].

A simple multi-fragment rendering technique is *depth peeling* [Eve01], which essentially renders the scene iteratively starting from the closest fragment layer and successively renders layers front-to-back. Advances approaches, which can peel multiple layers in a single render pass exist [BM08; Liu+09]. However, since it is an iterative approach, the performance constraints are prohibitive for rendering point clouds,

which can have very high depth complexity due to overlapping splats, especially in the case of outliers and high noise levels.

A-buffer (for *accumulation* buffer) methods [Car84] store *all* fragments of a pixel in a single render pass and are more closely related to the work presented in this chapter. Since there are currently no hardware implementations for A-buffers, all methods write the fragments to preallocated data structures in GPU memory, e. g., linked lists [Yan+10], fixed-size per-pixel arrays [Liu+10], or hash-based methods like the HA-buffer approach [LHL13]. As (noisy) point clouds have high depth complexity, the memory requirements for storing all fragments are extremely challenging and prohibitive for real-time rendering.

A-buffer

A more memory efficient variant of the A-buffer is the *k*-buffer, which only stores the *k best* fragments per pixel [Bav+07; JC99] (the meaning of *best* depends on the task, e. g., for transparency it would be the *k* closest fragments). Kim and Kye [KK21] proposed an A-buffer compatible *k*-buffer variant with fragment merging for order independent transparency. This approach, however, either stores a complete list of fragments and sorts the fragments for merging, which requires a lot of memory, or maintains a sorted list of fragments and merges new incoming fragments into the list at the correct positions. Inserting new fragments from an unordered fragment stream into a sorted list has an overall worst case complexity of $\mathcal{O}(n^2)$. Opposed to *k*-buffer algorithms, the proposed method can handle varying point densities. The approach only requires three render passes, like EWA, and realizes *on-the-fly* noise removal by identifying the closest surface above a minimal density threshold without any sorting of surfaces, fragments or splats. The hash-based data structure allows simultaneous rendering of the scene, merging of fragments and finding of the closest surface, and provides insertion of new fragments and retrieving the closest surface with $\mathcal{O}(1)$ runtime complexity.

7.3 PROBLEMS OF EWA WITH LARGE OUTLIERS

This section will briefly discuss the main problem of the Elliptical Weighted Average (EWA) surface splatting approach by Zwicker et al. [Zwi+01], as explained in section Section 2.4.3, when large outliers are present in the point cloud. Particularly, the analysis is based on the 3-pass implementation of EWA, which consists of visibility pass, accumulation pass and normalization pass. For clarity, the main equations of EWA are repeated here: Point clouds are defined as the sets

$$\mathcal{P} = \{(x_i, \mathbf{n}_i, r_i, \mathbf{a}_i)\}_{i=1}^N, \quad (7.1)$$

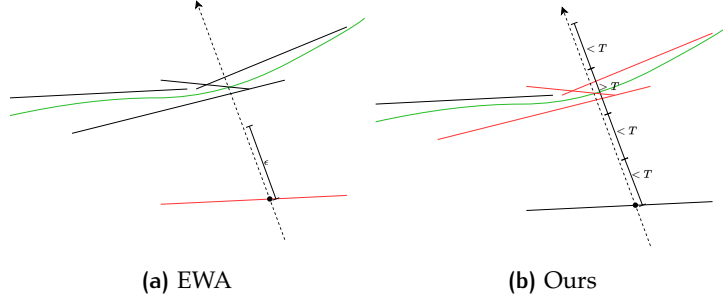


Figure 7.1: Comparison of the EWA implementation of [BK03] and the proposed bin based method. Pixel contributions for the viewing ray (dotted) are marked in red. The outlier splat is ignored with our method and only contributions close to the real surface (green) are considered for the averaging, since their bin passes the threshold T .

where each point has a position x_i , a normal n_i , a radius r_i and attributes a_i . EWA accumulates attributes of overlapping splats at pixel position (u, v) with the equation

$$a(u, v) = \frac{\sum_{i \in \mathcal{I}_\epsilon} w_i(\|q_i(u, v) - x_i\|) a_i}{\sum_{i \in \mathcal{I}_\epsilon} w_i(\|q_i(u, v) - x_i\|)}, \quad (7.2)$$

where $q_i(u, v)$ denotes the intersection of the viewing ray with the i -th splat and $\mathcal{I}_\epsilon$ is the set of splats that is at most ϵ away in viewing direction from the splat closest to the camera. The visibility pass computes that closest splat, while the accumulation pass computes the numerator and denominator of equation (7.2). Finally, the normalization pass computes the interpolated attribute $a(u, v)$. The weighting is done with a radial kernel function that falls to 0 toward the splat boundary, such as a Gaussian or the Wendland function [Wen95]

$$w_i(d) = \begin{cases} (1 - d/r_i)^4(4d/r_i + 1) & \text{if } d/r_i \leq 1 \\ 0 & \text{otherwise} \end{cases}. \quad (7.3)$$

The main problem of rendering noisy point clouds with EWA lies in the visibility pass. For instance, assuming Gaussian noise of the points around the true surface with outliers, it is not possible to set the surface thickness parameter ϵ to work in all cases. Setting $\epsilon = \infty$ would be required to prevent all extreme outliers from excluding splats near the surface to be excluded in the computations. This is exactly the situation when using real-world sensor data, as the noise level of, for instance, ToF cameras, depends on various aspects, such as the background light intensity, the surface reflectivity, and the scene geometry (see Section 2.1.2.6), and can not be captured with a global threshold. Moreover, in general, the surface thickness ϵ should be chosen as small as possible, so that distinct surfaces can be separated and farther surface layers cannot interfere with the front surface layer.

Figure 7.1(a) demonstrates this problem, i.e., the red outlier splat excludes the surface splats because the real surface is more than ϵ away.

A potential avenue to fix this issue is to combine thresholding in the normalization pass to discard pixels whose summed weights are smaller than a certain predefined value [DB07] or depth peeling [Eve01] to progressively fill in holes with valid surfaces from further behind, as this information has been skipped in the current accumulation pass. This approach is, however, very costly, as it requires re-rendering the scene multiple times.

7.4 PROPOSED METHOD

This section introduces the proposed *hashed, binned A-buffer* approach that solves the problems discussed in Section 7.3. The method is a direct extension of the 3-pass EWA implementation described in Section 2.4.3 and efficiently removes outliers when rendering noisy point clouds, without adding empty regions with missing pixels in the rendered image. First, the concept of binned A-buffers will be explained in Section 7.4.1. Afterward, an efficient GPU implementation of the approach is presented in Section 7.4.2. Finally, the per-pixel hashing technique to drastically increase memory efficiency will be explained in Section 7.4.3.

7.4.1 Binned A-buffer

As discussed in Section 7.3, the core of the problem of rendering noisy point clouds with outliers using traditional EWA splatting is the fact that only information about splats within ϵ depth distance to the points closest to the camera is retained after the visibility pass, and discarding outlier pixels requires re-rendering the scene or hole-filling. Therefore, this section proposes to generate *binned A-buffers* with equidistant bins that allow for a *parallel accumulation pass* along the viewing ray. Denoting the distance from the camera to the closest splat along a viewing ray at pixel (u, v) as $t_{\min}(u, v)$, the bin index of the camera ray intersection with a splat i at distance t_i is computed as

$$b(u, v, t_i) = \left\lfloor \frac{(t_i - t_{\min})}{\Delta} \right\rfloor, \quad (7.4)$$

where $\lfloor \cdot \rfloor$ denotes the floor operation and Δ is the bin size. Using this formulation, the binned A-buffer starts at the closest splat and uniformly extends in viewing ray direction, where the depth extend is limited by fixing a maximum bin count for now.

For each pixel and each bin, multiple values are accumulated into separate buffers: the splat weights $w_i(\|q_i(u, v) - x_i\|)$ corresponding

to the splat's intersection with the viewing ray segment, and the accumulated values $w_i(\|q_i(u, v) - x_i\|)a_i$ (see [equation \(7.2\)](#)). Compared to traditional [EWA](#), this means that there is an additional axis added to the accumulation buffers, depending now not only on the pixel coordinate (u, v) but also the bin index $b \in \mathbb{N}$. Denoting

$$\mathcal{I}(u, v, b) = \{i \mid b(u, v, t_i) = b\} \quad (7.5)$$

as the set of splat indices whose ray intersections lie in bin b , the accumulated values $\bar{a}(u, v, b)$ and accumulated weights $\bar{w}(u, v, b)$ in each bin can be defined as

$$\bar{a}(u, v, b) = \sum_{i \in \mathcal{I}(u, v, b)} w_i(\|q_i(u, v) - x_i\|)a_i \quad \text{and} \quad (7.6)$$

$$\bar{w}(u, v, b) = \sum_{i \in \mathcal{I}(u, v, b)} w_i(\|q_i(u, v) - x_i\|) \quad (7.7)$$

To filter out outliers, the closest bin $\hat{b}$, which contains a value larger than a predefined threshold T needs to be found, i. e.,

$$\hat{b}(u, v) = \min\{b \mid \bar{w}(u, v, b) > T\}. \quad (7.8)$$

This is analogous to the thresholding approach for [EWA](#) [[DBo7](#)], except that if the closest bin does not pass this threshold, the bins only need to be traversed further in depth-order instead of rendering the scene again. Pixels are discarded only, if *none* of the A-buffer bins contain sufficient (weighted) splats to reconstruct a surface. Otherwise, the closest A-buffer bin exceeding the weight threshold is taken as a surface intersection, and its attributes are normalized as

$$a(u, v) = \frac{\bar{a}(u, v, \hat{b}(u, v))}{\bar{w}(u, v, \hat{b}(u, v))}. \quad (7.9)$$

[Figure 7.1\(b\)](#) depicts the concept of the binned A-buffer.

An apparent issue in binning is the impact of the segmentation of the underlying data. Unfavorable locations of the bin borders, i. e., being directly on a surface, could lead to surfaces misses, because the highest density region is split into two bins. Experiments were conducted with an extension to the proposed method which uses an A-buffer with *overlapping* bins that can avoid this situation; however, the improvements were only marginal at the cost of decreased performance and memory efficiency. A detailed analysis of using overlapping bins can be found later in [Section 7.4.4](#).

7.4.2 GPU implementation

The method is implemented as a 3-pass approach using modern OpenGL. The first pass is a depth-only pass, where $t_{\min}(u, v)$ is computed with simple minimum blending and is equivalent to the visibility pass in [EWA](#) (see [Section 2.4.3](#)). The second pass contains the bulk

```

1  int  binIdx = clamp((t - t_min) / delta, 0, numArrayLayers);
2  ivec3 coord = ivec3(gl_FragCoord.xy, binIdx);
3  float dist  = length(splatCenter - viewRayIntersection);
4
5  float currWeight = wendland(dist, splatRadius);
6  vec3  currColor  = currWeight * splatColor;
7  float currDepth  = currWeight * splatCenter.z;
8
9  float prevWeight = imageAtomicAdd(weightTexture, coords,
    currWeight);
10 imageAtomicAdd(colorTexture, coords, currColor);
11 imageAtomicAdd(depthTexture, coords, currDepth);
12
13 if((prevWeight + currWeight) >= threshold){
14     minBinAboveTreshold = binIdx; // fragment output; min-
    blending activated
15 }

```

Listing 7.1: GLSL-style pseudocode for the A-buffer computation pass. In this example, two attributes are used, color and depth.

of computations. Here, the binned A-buffer is generated, and, in parallel, the smallest bin that exceeds the threshold is identified in a single pass. The per-pixel bins are implemented as a pre-allocated texture array of size $W \times H \times B$, where B is the number of bins. This means the maximum depth in the scene is limited to $t_{\max}(u, v) = t_{\min} + B\Delta$. As mentioned above, separate texture arrays for the weights and additional ones for each point attribute is needed.

As in GPU-based [EWA](#), the weight of the current splat w_{curr} is computed based on the view-space fragment position inside a fragment program, which are invoked in parallel for each fragment (even if the fragments belong to the same pixel). Using the image load/store functionality of OpenGL, these values can be directly added to the corresponding bin b of the texture array in a thread-safe manner by using the `imageAtomicAdd` function. This function also returns the previous bin value w_{prev} , i. e., the value that was inside the bin before the atomic operation. The minimal bin index currently above the threshold is determined via OpenGL's build-in min-blending. Note that since all weights are positive, the accumulated weight value is monotonically increasing for each bin. Thus, it is guaranteed that inside the fragment program the weight inside the bin *after* the atomic addition was executed is *at least* $w_{\text{prev}} + w_{\text{curr}}$. The monotonicity also ensures that bins cannot fall below the threshold anymore, and min-blending delivers the correct $\hat{b}$. The essence of a GLSL implementation of the proposed algorithm is given in [Listing 7.1](#).

After computing the binned A-buffer, the normalization pass consists of reading the weight and attribute buffers at the computed

```

1  uint hash(uint val) {
2      // 32-Bit Fibonacci hashing
3      return (val*2654435769) >> (32 - B);
4  }
5
6  uint binIdx      = int((t - tMin) / delta);
7  uint hashIdx     = hash(binIdx);
8  uint hashTableSize = pow(2, B);
9  uint emptyBin    = hashTableSize; // convention
10
11 // ... compute weights and weighted attributes as usual
12
13 // linear probing loop
14 for (int i = 0; i < hashTableSize; i++) {
15     uint currIdx = int(hashIdx + i) % hashTableSize;
16     ivec3 coords = ivec3(gl_FragCoord.xy, currIdx);
17     uint prev    = imageAtomicCompSwap(hashTableTexture,
18                                         coords, emptyBin, binIdx);
19
20     // if prev is ...
21     // ... emptyBin, the bin is empty
22     // ... binIdx, the bin contains values of the same surface
23     // ... anything else, a collision occurred
24     if(prev == emptyBin || prev == binIdx){
25         // 1. add weight and attributes to the cell at currIdx
26         // 2. write binIdx to frame buffer if weight is over
27         //    threshold
28         return;
29     }
30     // only reached if collision happened
31 }
32 // only reached if all bins are occupied

```

Listing 7.2: Hashing scheme with collision handling

minimal bin index. Like in traditional [EWA](#), this pass does not require a re-rendering of the scene. In case no bin above the threshold T was found (missed pixel), the last bin that contains any information or a potential background color can be rendered.

7.4.3 Spatial Hashing

The usage of a 3D texture array to represent the binned A-buffer introduces two major problems. First, the maximum depth of the scene is limited to $t_{\max}(u, v) = t_{\min} + B\Delta$ for B bins in viewing direction. This is a problem especially for scenes with large z-extend, but relatively low noise, for which smaller values for Δ are more optimal, because the necessary texture memory may exceed GPU limits. Note, that in the pseudocode in [Listing 7.1](#) fragments beyond $t_{\max}(u, v)$ are rendered into the last bin by clamping values to the maximum possible bin

index. The second problem is that most bins are actually unused, since in general a large amount of space in the scene is empty. This leads to a lot of wasted memory, and occupied bins are fragmented across large memory locations, potentially yielding poor cache locality.

To fix these issues, a hashing scheme that utilizes the available memory more efficiently is introduced. Conveniently, the hashing scheme also makes the memory requirements independent of the depth extend of the scene.

A hashing function that maps any positive integer into the integer range $[0, 2^B - 1]$ is used to compute the bin index into the texture arrays. For the proposed method in this chapter, Fibonacci hashing [Don98] is chosen. However, other hashing functions, which potentially do not require a power of 2 sized array, could be chosen as well. The input of the hash function is the computed bin index b . Collisions are resolved with linear probing, which is efficiently implemented with the use of the `atomicCompSwap` function.

The fragment shader pseudocode for the hash table computation including linear probing can be found in Listing 7.2. The bin index also has to be translated to a hash table index in the normalization pass, with the same collision handling. While in the worst case this could lead to looping over all hash table indices, in practice, the computational overhead compared to the approach without hashing is negligible and the better cache locality due to smaller arrays even improves performance.

7.4.4 Handling surfaces at bin borders

As briefly mentioned at the end of Section 7.4.1, a common problem in binning is the impact of the segmentation on the underlying data. In the case of the binned A-buffer algorithm, splats corresponding to the surface of interest might be accumulated into two neighboring bins that both may stay below the threshold T , leading to a surface miss; see Figure 7.2(a). Unfortunately, using the efficient 3-pass approach, this issue cannot be solved by just adding the accumulated values from both bins during the normalization pass, since only the closest bin *above* the threshold is known, i. e., the only option would be a costly line search through the whole A-buffer.

A simple way to overcome this apparent issue is the introduction of a second *virtual* A-buffer, whose bins are shifted by half the bin size to the back (see Figure 7.2(b)), and add incoming splats to both buffers. By interleaving both virtual A-buffers into a single global buffer with now overlapping bins, i. e., the global buffer alternates between bins from the primary and shifted buffers, the index of the closest bin above the threshold can then be efficiently computed as described in the previous sections. However, this approach requires twice the number of atomic adds as the disjoint approach, which leads

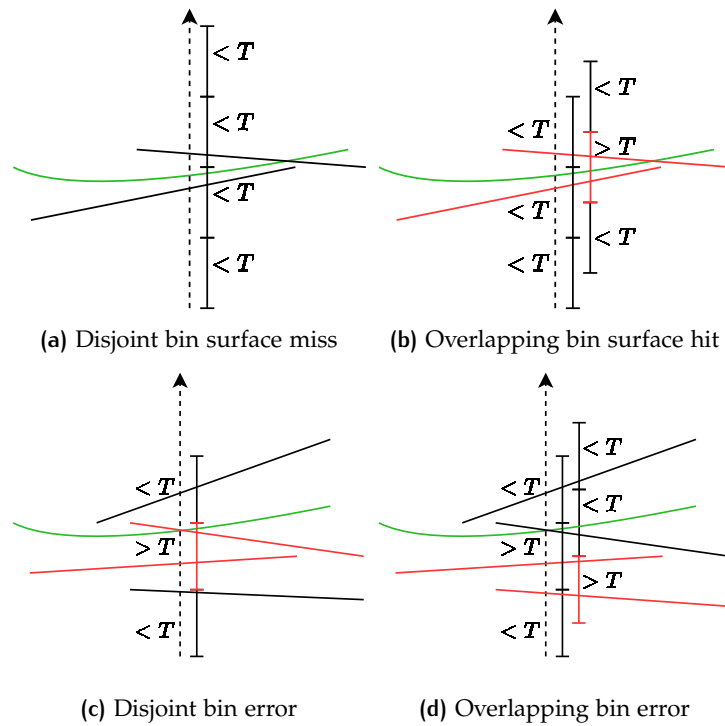


Figure 7.2: Different situations to illustrate the advantages and disadvantages of using overlapping bins.

(a) Since the bin border is right at the surface, the two splats are put into different bins and the threshold is not reached, resulting in a surface miss. (b) Overlapping bins fix this because bin borders completely lie inside the next bin. (c), (d) If there is already a surface hit when using disjoint bins, overlapping bins may result in a larger bias towards the camera.

to a significant performance reduction. Additionally, this effectively doubles the memory consumption to avoid hash collisions.

For a given fixed threshold and a single surface, if there is already a bin found by the standard (disjoint) bin method, the overlapping bin variant would not improve the error at that pixel (see [Figures 7.2\(c\)](#) and [7.2\(d\)](#)). The reason for this is, that the primary A-buffer (i. e., the non-shifted one) of the overlapping bin approach is identical to the single A-buffer when using disjoint bins. Since the algorithm selects the *closest* bin over the threshold, it can *at best* select the same bin for both variants, or worse, a bin that is closer to the camera when using overlapping bins. The latter necessarily introduces a larger error, because the method is already slightly biased toward the front, due to the discretizations, even for optimally chosen thresholds. This means that the use of overlapping bins is in general not worth the increased cost. Ablation experiments that support this analysis can be found in [Section 7.5.6](#). For this reason, the other of following experiments use disjoint bins as described in [Sections 7.4.1](#) and [7.4.3](#) instead.

7.5 EXPERIMENTS AND EVALUATION

The effectiveness of the proposed binned splatting approach was evaluated on a variety of scenes and compared against naive *opaque* splatting and the GPU-based [EWA](#) splatting approach by Botsch and Kobbelt [[BK03](#)]. In both [EWA](#) splatting and binned splatting, the depth, normals, and color (if available), were accumulated. If no color is available in a scene, a white material with simple Phong shading is applied. If not stated differently, [EWA](#)’s surface thickness parameter and the bin size are set equal to 1 cm, i. e., $\epsilon = \Delta = 0.01$, while the threshold T is manually adjusted according to the noise level and point density.

Unfortunately, there is no publicly available code base for Diankov and Bajcsy [[DB07](#)]. Moreover, their method also applies hole-filling, i. e., a comparison of the rendering results would be biased, and a runtime comparison gives little insight, as the method applies 3-5 full [EWA](#) passes, i. e., 9-15 render passes in total, compared to only needing 3 passes in the binned splatting method.

Conceptually, the experiment setup is similar to Preiner et al. [[PJW12](#)] and Diankov and Bajcsy [[DB07](#)] and assumes that the ground truth surface is captured by multiple RGB-D observations with known camera pose, see [Figure 7.3](#) for a visualization of the setup. This scenario emulates real-world scene acquisition using hand-held range camera or RGB-D cameras. The noisy point sets from all camera observations are transformed into world space and concatenated to a single point cloud, which is rendered from novel camera positions. No preprocessing is applied to the point clouds, except the computation

Experiment setup

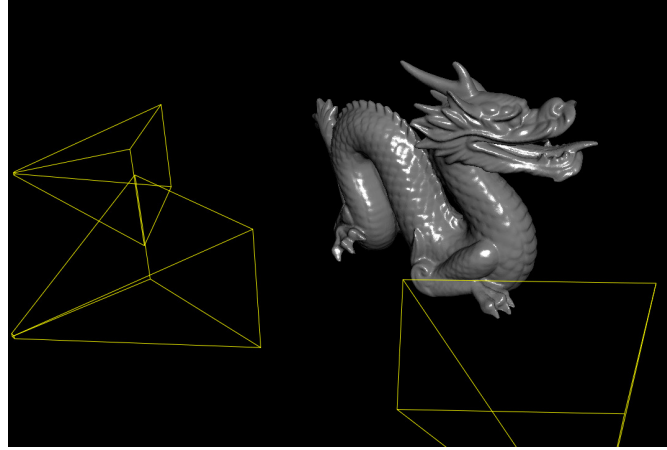


Figure 7.3: General experiment setup. Multiple depth cameras with known positions (yellow) capture depth maps, which are projected back into world space to get the point clouds to be rendered.

of point normals using a simple cross-product in image space, applied to (auxiliary) bilaterally filtered range images. All compared methods receive the exact same splat geometry as their input and render normal-aligned splats. The following datasets and scenes are used for comparison:

FIREPLACE ROOM [McG17] This synthetic scene provides ground truth depth. The point cloud is generated from 10 depth views rendered from different distances and angles, focusing a horse statue, and consists of approximately 3.5 million points. Poisson noise approximated as Gaussian noise was added. The standard deviation depends on the depth z during the point cloud measurement and a noise level parameter k and is computed as $\sigma(z) = k(z^2/1000)$; see Figure 7.4.

TABLE SCENE This synthetic scene was generating using the ToF simulator by Lambers et al. [LHK15] to accurately simulate four RGB-D images and contains approximately 300,000 points; see Figure 7.5.

ICL-NUIM The *livingroom1* and *office1* sequences were taken from the augmented ICL-NUIM dataset [CZK15; Han+14]. The point clouds were generated from 7 RGB-D input images, 10 frames apart from each other in the original video stream, and contain approximately 1.4 million points for the *livingroom1* and 1.8 million for the *office1* scene. The ICL-NUIM dataset was generated by the authors with a simulator emulating a structured light Microsoft Kinect v1 camera. Ground truth poses and color images for evaluation are taken from other frames not used in the point cloud; see Figure 7.7.

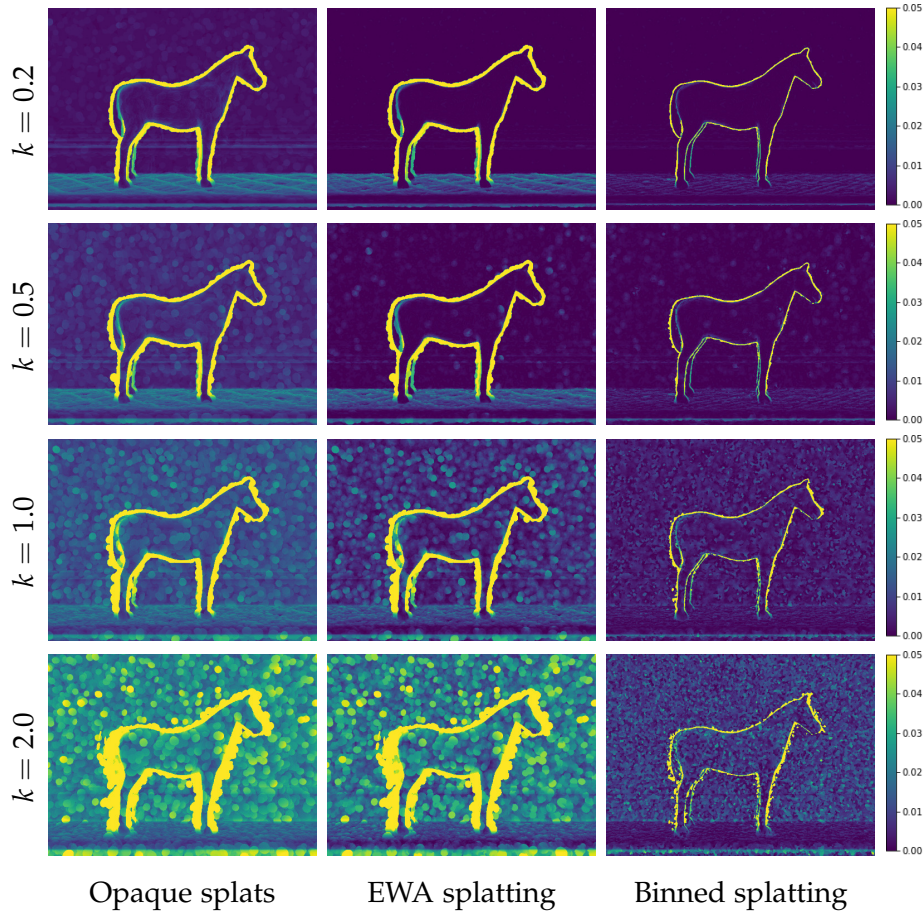


Figure 7.4: Difference to ground truth depth for different noise levels in the *fireplace room* scene for different noise levels k . Errors outside of the color bar range are clamped for visualization purposes.

MEMORIAL This real-world dataset contains 10 images for approximately 0.64 million points from a RGB-D outdoor sequence captured with a ToF-based Kinect v2 camera under strong ambient lighting conditions, yielding very noisy range data; see Figure 7.6.

STANFORD DRAGON [CLg6] A point cloud with approximately 1.5 million points, augmented by a randomly sampled 125,000 points in its bounding box to simulate a very large amount of outliers; see Figure 7.8.

7.5.1 Depth reconstruction accuracy

The depth reconstruction accuracy was quantitatively evaluated using the *fireplace room* and *table scene* datasets. In addition, it was qualitatively evaluated using the *memorial* dataset.

Table 7.1 and Figure 7.4 state the quantitative results of the depth reconstruction for the *fireplace room* scene; Figure 7.5, bottom row gives

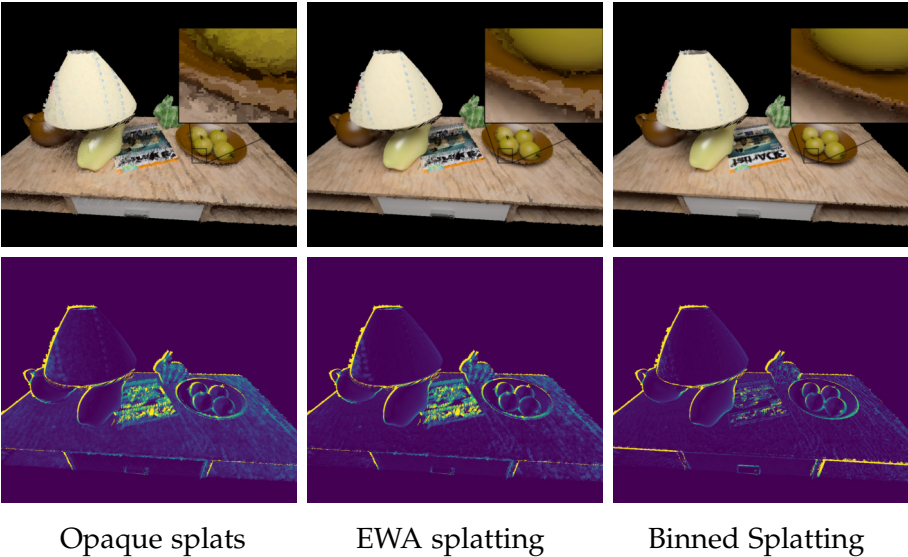


Figure 7.5: Color (top row) and depth (bottom row) reconstruction results for the synthetic *table scene* sequence.

	$k = 0.2$		$k = 0.5$		$k = 1.0$		$k = 2.0$	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Opaque	0.78	1.74	1.13	1.96	1.80	2.55	3.20	4.01
EWA	0.47	1.61	0.59	1.76	1.29	2.31	2.86	3.82
Binned	0.17	0.91	0.27	1.02	0.61	1.24	1.20	1.86

Table 7.1: Quantitative results for the *fireplace room* scene for different noise levels. All errors are given in cm.

the quantitative results for the *table scene* dataset. The error maps display the absolute depth error between the reconstruction results obtained from rendering the point cloud and ground truth mesh. The depth error maps show that, in general, the binned A-buffer approach significantly increases the depth reconstruction quality, both in regions with homogeneous depth and at jumping edges with wrong normal alignment. This gets especially apparent at the depth discontinuity between the horse statue and the background wall in Figure 7.4, leading to a similar effect as achieved by the introduction of clipped splats in Botsch et al. [BSK04] and Pauly et al. [Pau+03], albeit without the need to precompute the clipping lines. Since the distance between the horse figure and the wall is larger than the surface thickness ($\epsilon = \Delta = 0.01$), EWA splatting renders splat parts that protrude the horse’s boundary, while the binned A-buffer splatting approach skips them to a large degree, as the splat density in these regions is low. Increasing noise levels produces more outliers farther away from the ground truth surface, which manifests in isolated splats with large errors when using traditional EWA splatting, while the binned A-buffer approach exhibits significantly reduced errors, which is underlined in the mean absolute error (MAE) and RMSE values in Table 7.1. Similar effects can be seen at the fruit bowl in Figure 7.5. Moreover, strong reduced depth errors on the magazine can be observed, which is mainly due to the higher noise level on the dark scene region (see Section 2.1.2.6).

However, at the edge of the table in the *table scene*, an increase of errors can be seen. This is mainly due to the usage of a *global* threshold parameter T that generates either holes at the edge of the table or does not sufficiently suppress the outliers. In this particular case the algorithm fails to find a bin at the table edge that is above the threshold and instead renders a bin further from the camera in the background of the scene, resulting in a large error.

Considering the real-world *memorial* dataset in Figure 7.6, bottom row, the binned A-buffer approach again achieves a clearly improved depth quality in homogeneous regions as well as at edges.

7.5.2 Color reconstruction quality

The color reconstruction accuracy is evaluated on the *table scene*, the *memorial* dataset, as well as the two ICL-NUIM scenes. As seen from the results in Table 7.2, the binned A-buffer splatting consistently outperforms the other two approaches.

Figure 7.7 depicts the visual color reconstruction of the two ICL-NUIM scenes in comparison with a ground truth pose. While the noise in this dataset is subtle, outliers can be seen at the border of the vase and the armrest of the chair in the *livingroom1* scene and at the cables in the *office1* scene, which are nicely removed by the binning method.

Method	livingroom1	office1	table
Basic	22.89	23.57	25.05
EWA	26.73	27.15	25.99
Binned	27.74	28.24	27.73

Table 7.2: Quantitative color reconstruction results (PSNR) on the ICL-NUIM scenes.

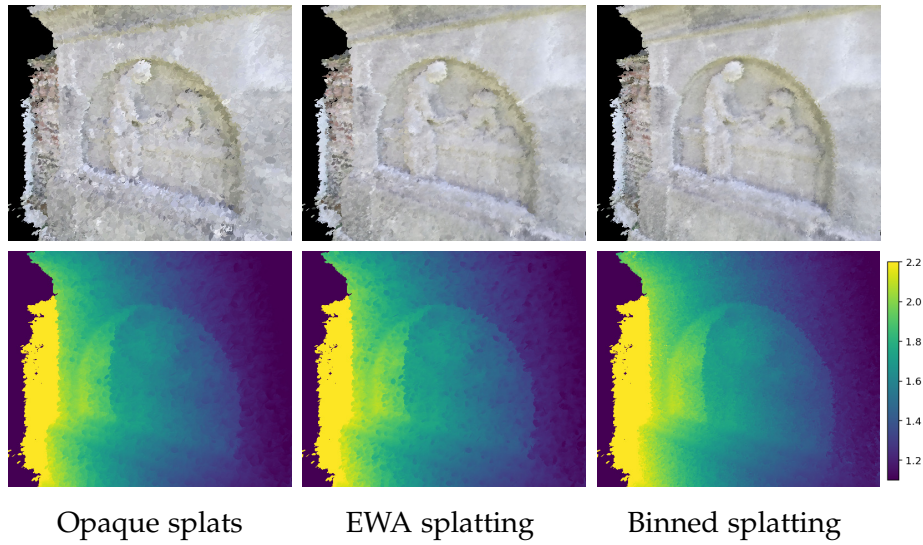


Figure 7.6: Reconstruction results for color (top row) and depth (bottom row) on the real world *memorial* data set. Due to the high noise we set $\Delta = \epsilon = 0.05m$

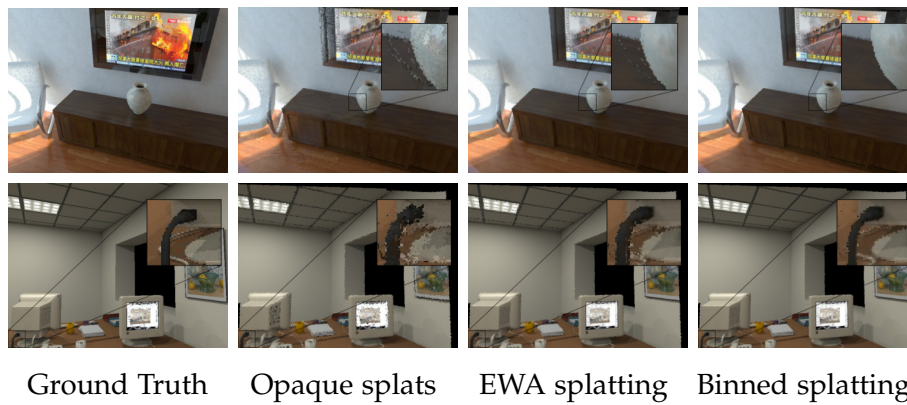


Figure 7.7: Results on the augmented ICL-NUIM dataset [CZK15]. Top row: *livingroom1* sequence. Bottom row: *office1* sequence.

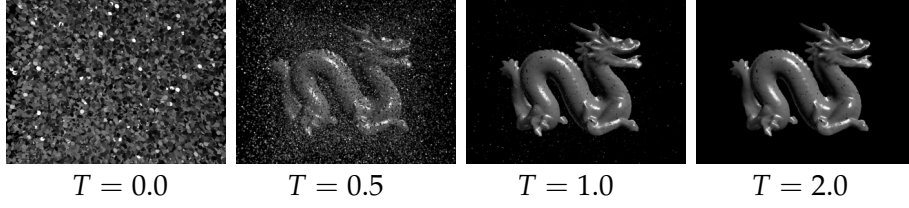


Figure 7.8: *Stanford Dragon* with extreme outliers and different thresholds ($\Delta = 0.01$)

In these regions, EWA splatting exhibits strong dilation-like effects that considerably reduce the color reconstruction quality. The *table scene* dataset contains artifacts commonly seen in ToF depth data, such as flying pixels in darker regions (magazine) and at depth discontinuities (fruit bowl); see Figure 7.5. The binned A-buffer approach clearly exceeds EWA’s quality in these image regions by removing all flying pixels and picking the proper surface color, thus making the text legible. In the real-world *memorial* dataset, depicted in Figure 7.6, similar effects can be seen also in homogeneous regions. While EWA yields many areas that are visually similar to opaque splatting, the binned A-buffer approach further removes these unwanted artifacts.

7.5.3 High noise and outlier levels

The robustness of the binned A-buffer approach against high noise levels and strong outliers is evaluated visually on the *dragon* and *memorial* datasets. The *dragon* dataset contains extreme outliers in a cube around the actual object of interest and showcases the impact of the threshold parameter T on the rendering scheme. For $T = 0$, the binned A-buffer approach is equivalent to traditional EWA splatting if $\epsilon = \Delta$ (which is always the case in this chapter). Figure 7.8 shows that the binned A-buffer method can nicely extract the object’s surface by adapting the threshold T , i. e., for increasing T the outliers are filtered out and only the more densely sampled surface is rendered. Note, that all 4 images are the result of a single iteration of the proposed 3-pass approach. The only difference is that the data for the final image is extracted from different bins of the A-buffer.

Considering the real-world *memorial* dataset that comprises very high noise level under daylight conditions in Figure 7.6, the superiority of the binned A-buffer approach compared to EWA can be clearly confirmed, both in terms of depth and color construction quality. Note, that in contrast to most other experiments, due to high noise in this scene, the surface thickness and the bin size is adapted to $\epsilon = \Delta = 0.05$ in this scene, while the threshold remains at 1.

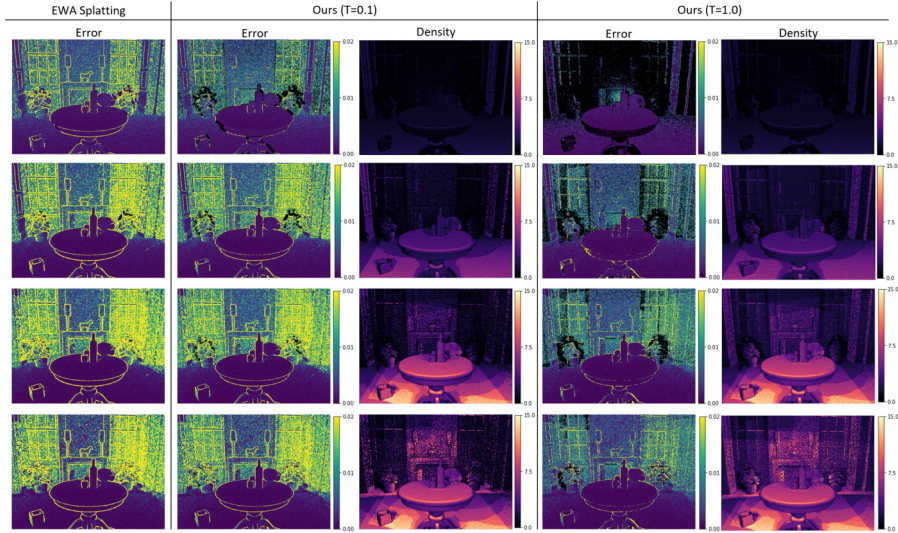


Figure 7.9: Evaluation on the *fireplace room* scene with different point densities and thresholds. Each row adds points from additional capturing positions to the point cloud. The density visualization refers to the weight in the bin selected by the algorithm. Note the increased noise in the background, resulting from the distance-dependent noise model. Error images are per-pixel differences to ground truth depth. Missing pixels, i. e., pixels where no bin over the threshold was found by the algorithm, are set to black.

7.5.4 Robustness to varying density

The binned A-buffer method assumes that the point cloud has higher density close to the true surface. The (global) threshold parameter T defines the minimal point density, i. e., any weighted contribution below T is suppressed, which leads to issues on point clouds where the density varies spatially. Moreover, a larger number of splats per bin improves the denoising effect, i. e., detected surfaces for large T are more reliable.

The experiment setup (see Figure 7.3) already induces a spatially varying point density and noise level in all scenes, since the points originate from different, overlapping capturing positions, and the noise model is distance dependent. To further evaluate the robustness of the method, it is also applied on a challenging scene with highly varying density and noise distributions (see Figure 7.9).

As expected, higher thresholds filter out points in low-density regions. In general, increasing the threshold results in only higher-quality bins to be retained, which can be seen especially in the first row of Figure 7.9, where only pixels with low errors are present for $T = 1$. A higher density (bottom rows) allows the thresholds to be set higher, while still retaining a hole-free image with low errors. Moreover, the higher noise level toward the background results in a lower density per bin, which means more points are required here for good

Scene	#Points	EWA	Ours
fireplace Room	3.5M	92	64
livingroom1	1.4M	171	102
office1	1.8M	178	118
table scene	0.3M	380	260

Table 7.3: Framerate in different scenes. The value in parentheses are the FPS when using overlapping bins in our method.

Hash table size	FPS	Memory overhead
2^3	115	+98 MiB
2^4	95	+294 MiB
2^5	56	+700 MiB
2^6	34	+1471 MiB

Table 7.4: Measured memory overhead over EWA and FPS of our method on the *Memorial* dataset. EWA renders the scene at 179 FPS.

results. In any case, even for $T = 0.1$, the approach provides better results than [EWA](#).

7.5.5 Performance

All experiments were performed on a machine with an Intel i7-8600K processor and a GeForce 1080 Ti GPU. The method achieves real-time framerates for all shown scenes and experiments in this chapter (see [Table 7.3](#)). Since the contributions to all surfaces along a viewing ray are accumulated at once, the binned A-buffer approach has a computational and memory overhead compared to [EWA](#) splatting on all scenes. The [EWA](#) splatting implementation used for comparison also uses hardware blending for accumulation, which in general is faster than `imageAtomicAdd` operations necessary for the A-buffer implementation. In general, the memory overhead depends on the chosen hash table size, which cannot be too small because it both leads to more hash collisions and could result in overflow if the hash table is smaller than the amount of surfaces in the current viewing direction. However, if the hash table size is too large, the performance of the method also degrades, which might be due to worse cache locality of the texture accesses.

Exemplary samples of the runtime performance as well as memory overhead were measured for different hash table sizes on the *memorial* dataset, which is quite challenging for the proposed algorithm, since the high noise and many outliers result in write accesses to many different bins per pixel. The scene contains 654,083 points that are

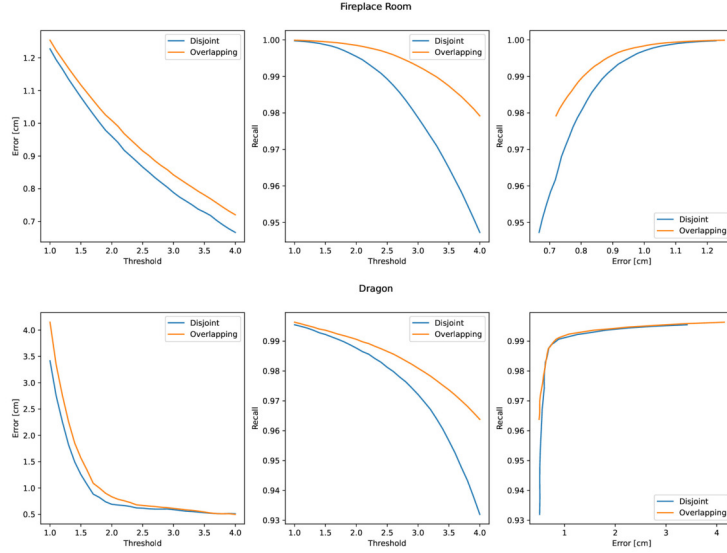


Figure 7.10: Error/recall trade-off of using disjoint overlapping bins with different thresholds.

rendered at a resolution of 1024×848 , which is twice the resolution of the input images. The results can be found in Table 7.4. A hash table size below 2^3 results in memory overflow and artifacts occur. Note, that sizes other than a power of two would be possible if a different hash function was used.

7.5.6 Ablations with overlapping bins

To validate the analysis in Section 7.4.4, experiments comparing the use of disjoint and overlapping bins and the behavior under different thresholds were conducted. The variants were run on the *Stanford dragon* and *fireplace room* datasets with noise level $k = 1$ and varying thresholds T , and the depth reconstruction accuracy was compared to the ground truth depth. The experiments compute the RMSE, where the average is only taken over pixels where a bin was found, and the *recall*, which is here defined as the ratio of rendered pixels and pixels in the ground truth image, in order to identify the amount of surface misses.

The results can be seen in Figures 7.10 and 7.11. With increasing thresholds only higher-quality bins (i.e., with higher density) are selected, which means that the average error decreases. As hypothesized in Section 7.4.4, using overlapping bins leads to a higher error compared to disjoint bins. A larger threshold also introduces more holes, i.e., a lower recall. Overlapping bins show a recall improvement compared to disjoint bins, since situations like in Figure 7.2(a) can be avoided. The plots in the rightmost column of Figure 7.10 clearly show an error/recall trade-off when choosing a threshold. They also show

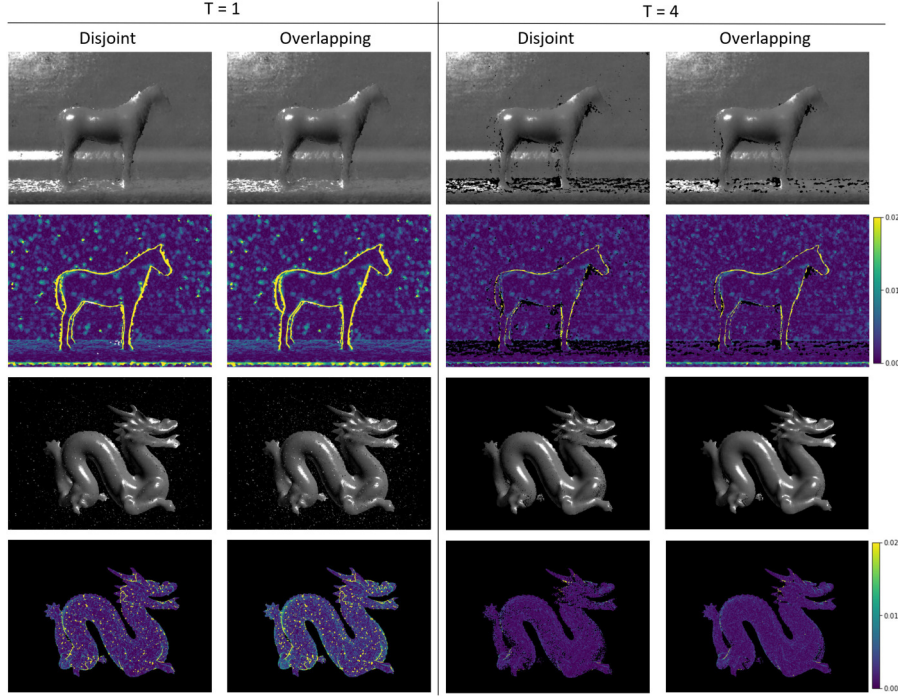


Figure 7.11: Example rendering comparisons using disjoint and overlapping bins. Pixels that do not contribute to the error calculation are rendered black.

that in order to reach the same recall as with overlapping bins, only a slight error penalty (≈ 0.1 cm on *fireplace room* for 90% recall) has to be sacrificed when choosing disjoint bins and a lower threshold instead. Conversely, for the same error, a small decrease in recall ($\approx 1\%$ for an error of 7.25 cm) must be taken into account. While this suggests a slight advantage of overlapping bins when using optimal thresholds, the choice between the two must also take the decreased framerate (up to half the framerate compared to disjoint bins in the experiments) and higher memory consumption into account, which is in general not worth it for the marginal improvements.

7.6 CONCLUSIONS AND LIMITATIONS

The main benefits of the binned A-buffer method regarding the quality of the rendered color and depth images are explicitly prominent for noisy (real-world) point clouds with strong outliers. The method can filter strong noise of surface splats and suppress large amounts of outliers. Moreover, it can suppress the dilation-like halo effect caused by splats at depth discontinuities by reducing the visibility of individual splats that protrude the contour of the front object.

The hashed, binned A-buffer algorithm implementation is efficient in terms of memory footprint and runtime performance. Still, there is

Future work

room for improvement, mainly in terms of the selection of the threshold value T . As discussed in [Section 7.5.4](#), in certain constellations there is no single global threshold that fits all local needs. Therefore, a theoretical investigation of the statistical relations between noise level, outlier density, and surface splat density aiming at an automated adaptation of T is a worthwhile future research objective.

This dissertation made various improvements to imaging pipelines. The central contributions can be split into the two main categories of end-to-end co-design of sensor parameters and high-level tasks to improve the imaging pipeline as a whole, and the handling and improvement of imperfect data arising from measurement errors.

The thesis presented a novel approach to jointly optimize the physical layout of pixels on a sensor and a task-specific neural network, by developing a differentiable pixel layout parameterization. The method can be integrated into existing deep learning pipelines and only requires standard high-resolution images as input for the differentiable sensor simulation. The proposed rectangular and curvilinear layouts outperform standard uniform pixel grids in all evaluated classification and segmentation tasks in autonomous driving, particularly under pixel budget constraints.

*End-to-end sensor
optimization*

Similarly, a method to learn color filters together with neural networks for specific tasks was proposed. In contrast to the spatial layout case, however, experiments on semantic segmentation for autonomous driving did not show a statistically significant benefit over hand-crafted filters. The discrepancy between this negative result and the positive results for learned sensor layouts indicates that not all sensor parameters are equally important for all tasks. The sensor layout optimization placed more pixels in areas of high information density as required by the downstream network. This assumes that such a *spatial bias* of information content even exists in the dataset, and more evenly distributed objects may not benefit as much from a learned layout.

A plausible hypothesis for the lack of improvement from learned color filters is that there is insufficient *spectral bias* in the autonomous driving datasets used for evaluation. In the investigated setting, the segmentation networks focus more on shape and texture than spectral differences, which is why they benefit more from increased resolution in challenging regions. It should be emphasized, that these negative results most likely do not apply to other datasets with more spectral bias.

Future work could evaluate the methods on other task and datasets, and could also include combinations of learned pixel layouts and color filters, which are straightforward to implement in the proposed frameworks. Additionally, both methods can benefit from a tighter integration of physical constraints and design restrictions on both the learned layouts and learned filters. Although recent work has shown that the learned rectangular and curvilinear layouts can be

built by developing hardware designs [Cho+24; Zha+25], more complex layout parameterizations likely require more constraints to be manufacturable.

The second main line of contributions in this dissertation concerns the handling of imperfect data produced by sensors. A provably convergent algorithmic scheme based on the proximal gradient method was proposed that combines classical variational methods with powerful learned priors to solve inverse problems. This combines the flexibility of plug-and-play schemes that do not require retraining a network for every task, with theoretical guarantees of energy minimization methods, which are often required in safety-critical settings. However, the method is limited to smooth energies. An extension to the non-smooth case and an incorporation in more complex schemes like ADMM could be possible research directions.

Handling imperfect data

Active depth sensors under poor lighting conditions often generate noisy, incomplete and outlier-prone measurements that must be processed before they can be reliably used for higher-level vision tasks. A generic depth enhancement framework based on normal consistency was proposed to enhance depth images by enforcing geometric coherence between depth and surface normals, improving depth image quality for a variety of depth enhancement tasks, i. e., denoising, super-resolution, and deblurring.

Finally, a point-based rendering method that can directly visualize very noisy point clouds with heavy outliers was developed. The method includes a per-pixel binned A-buffer data structure that includes a hashing scheme for memory efficiency. It successfully suppresses outliers, such as flying pixels, while still reconstructing visually plausible surfaces. This is particularly useful for the interactive inspection and debugging of 3D data from ToF cameras and similar sensors. The method involves a fixed global threshold parameter that is not optimal for all areas of a scene. Future work could investigate methods to automatically adapt this parameter, ideally per-pixel.

BIBLIOGRAPHY

- [AEB06] M. Aharon, M. Elad, and A. Bruckstein. “K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation.” In: *IEEE Transactions on Signal Processing* 54.11 (2006), pp. 4311–4322 (cit. on p. 42).
- [AR20] M. Aladem and S. A. Rawashdeh. “A single-stream segmentation and depth prediction CNN for autonomous driving.” In: *IEEE Intelligent Systems* 36.4 (2020), pp. 79–85 (cit. on p. 89).
- [AT11] E. Allen and S. Triantaphillidou. *The Manual of Photography*. 10th ed. Elsevier, 2011, p. 111 (cit. on p. 11).
- [Ave+20] P. H. C. Avelar, A. R. Tavares, T. L. T. da Silveira, C. R. Jung, and L. C. Lamb. “Superpixel Image Classification with Graph Attention Networks.” In: *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images*. 2020, pp. 203–209 (cit. on p. 50).
- [BKH16] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer normalization*. 2016. arXiv: 1607.06450 [stat.ML] (cit. on p. 28).
- [Bav+07] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. Comba, and C. T. Silva. “Multi-fragment effects on the GPU using the k-buffer.” In: *Proceedings of the 2007 symposium on Interactive 3D graphics and games*. 2007, pp. 97–104 (cit. on p. 109).
- [BM08] L. Bavoil and K. Myers. *Order independent transparency with dual depth peeling*. Tech. rep. Nvidia, 2008 (cit. on p. 108).
- [Bay76] B. E. Bayer. “Color imaging array.” U.S. pat. 3971065A. E. K. Co. July 20, 1976 (cit. on p. 18).
- [BLW07] M. Ben-Ezra, Z. Lin, and B. Wilburn. “Penrose pixels super-resolution in the detector layout domain.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2007, pp. 1–8 (cit. on p. 49).
- [BB18] M. Benning and M. Burger. “Modern regularization methods for inverse problems.” In: *Acta numerica* 27 (2018), pp. 1–111 (cit. on p. 81).
- [Bie+07] A. Bienert, S. Scheller, E. Keane, F. Mohan, and C. Nugent. “Tree detection and diameter estimations by analysis of forest terrestrial laserscanner point clouds.” In: *ISPRS Workshop on Laser Scanning*. Vol. 36. IAPRS Espoo, Finland. 2007, pp. 50–55 (cit. on pp. 43, 105).

- [Bjo+18] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger. "Understanding batch normalization." In: *Advances in Neural Information Processing Systems* 31 (2018) (cit. on p. 28).
- [Bot+05] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. "High-quality surface splatting on today's GPUs." In: *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005*. IEEE. 2005, pp. 17–141 (cit. on pp. 45, 46, 106, 107).
- [BK03] M. Botsch and L. Kobbelt. "High-quality point-based rendering on modern GPUs." In: *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. IEEE. 2003, pp. 335–343 (cit. on pp. 110, 117).
- [BSK04] M. Botsch, M. Spornat, and L. Kobbelt. "Phong splatting." In: *Proceedings of the First Eurographics conference on Point-Based Graphics*. 2004, pp. 25–32 (cit. on pp. 45, 106–108, 121).
- [BA06] J. Brauers and T. Aach. "A color filter array based multispectral camera." In: *12. Workshop Farbbildverarbeitung*. Ilmenau. 2006, pp. 5–6 (cit. on p. 75).
- [BLK17] D. Bulczak, M. Lambers, and A. Kolb. "Quantified, interactive simulation of AMCW ToF camera including multipath effects." In: *Sensors* 18.1 (2017), p. 13 (cit. on p. 22).
- [Car84] L. Carpenter. "The A-buffer, an antialiased hidden surface method." In: *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 1984, pp. 103–108 (cit. on p. 109).
- [Cha16] A. Chakrabarti. "Learning sensor multiplexing design through back-propagation." In: *Advances in Neural Information Processing Systems* 29 (2016) (cit. on pp. 69, 70).
- [CP11] A. Chambolle and T. Pock. "A first-order primal-dual algorithm for convex problems with applications to imaging." In: *Journal of Mathematical Imaging and Vision* 40.1 (2011), pp. 120–145 (cit. on p. 42).
- [CWE16] S. H. Chan, X. Wang, and O. A. Elgandy. "Plug-and-play ADMM for image restoration: Fixed-point convergence and applications." In: *IEEE Transactions on Computational Imaging* 3.1 (2016), pp. 84–98 (cit. on p. 82).
- [CW19] J. Chang and G. Wetzstein. "Deep Optics for Monocular Depth Estimation and 3D Object Detection." In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2019, pp. 10192–10201 (cit. on pp. 47, 50, 69).

- [Cha+18] J. Chang, V. Sitzmann, X. Dun, W. Heidrich, and G. Wetzstein. “Hybrid optical-electronic convolutional neural networks with optimized diffractive optics for image classification.” In: *Scientific Reports* 8 (2018) (cit. on pp. 47, 50, 69).
- [Cha+94] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud. “Two deterministic half-quadratic regularization algorithms for computed imaging.” In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. Vol. 2. IEEE. 1994, pp. 168–172 (cit. on p. 96).
- [Che+17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 40.4 (2017), pp. 834–848 (cit. on p. 75).
- [Che+18] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. “Encoder-decoder with atrous separable convolution for semantic image segmentation.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 801–818 (cit. on p. 64).
- [CRP14] Y. Chen, R. Ranftl, and T. Pock. “Insights into analysis operator learning: From patch-based sparse models to higher order MRFs.” In: *IEEE Transactions on Image Processing* 23.3 (2014), pp. 1060–1072 (cit. on p. 42).
- [CYP15] Y. Chen, W. Yu, and T. Pock. “On learning optimized reaction diffusion processes for effective image restoration.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 5261–5269 (cit. on p. 42).
- [Cho+14] J. Cho, J. Choi, S.-J. Kim, S. Park, J. Shin, J. D. Kim, and E. Yoon. “A 3-D camera with adaptable background light suppression using pixel-binning and super-resolution.” In: *IEEE Journal of Solid-State Circuits* 49.10 (2014), pp. 2319–2332 (cit. on p. 51).
- [CZK15] S. Choi, Q.-Y. Zhou, and V. Koltun. “Robust Reconstruction of Indoor Scenes.” In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 118, 122).
- [Cho+24] B. Choubey, H. Sommerhoff, M. Moeller, and A. Kolb. “Variable layout CMOS pixels for end-to-end learning in task specific image sensors.” In: *International Conference on AI Circuits and Systems (AICAS)*. IEEE. 2024, pp. 482–486 (cit. on pp. 48, 49, 51, 53, 54, 69, 130).

- [Chu+21] I. Chugunov, S.-H. Baek, Q. Fu, W. Heidrich, and F. Heide. “Mask-ToF: Learning Microlens Masks for Flying Pixel Correction in Time-of-Flight Imaging.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 9116–9126 (cit. on pp. 47, 50, 69).
- [Cor+16] 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.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 61, 62, 65).
- [CL96] B. Curless and M. Levoy. “A volumetric method for building complex models from range images.” In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 303–312 (cit. on p. 119).
- [Dai+17] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. “Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration.” In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 1 (cit. on p. 89).
- [Dai+20] P. Dai, Y. Zhang, Z. Li, S. Liu, and B. Zeng. “Neural point cloud rendering via multi-plane projection.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 7830–7839 (cit. on p. 108).
- [DKE10] A. Danielyan, V. Katkovnik, and K. Egiazarian. “Image deblurring by augmented Lagrangian with BM3D frame prior.” In: *Workshop on Information Theoretic Methods in Science and Engineering*. Vol. 1. 2010, p. 1 (cit. on pp. 42, 82).
- [Den+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A large-scale hierarchical image database.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255 (cit. on pp. 61, 66).
- [Den12] L. Deng. “The mnist database of handwritten digit images for machine learning research.” In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142 (cit. on pp. 59, 61).
- [Dia+21] S. Diamond, V. Sitzmann, F. Julca-Aguilar, S. Boyd, G. Wetzstein, and F. Heide. “Dirty pixels: Towards end-to-end image processing and perception.” In: *ACM Transactions on Graphics (TOG)* 40.3 (2021), pp. 1–15 (cit. on pp. 42, 69, 73).

- [DB07] R. Diankov and R. Bajcsy. “Real-time adaptive point splatting for noisy point clouds.” In: *International Conference on Computer Graphics Theory and Applications* 7 (2007), pp. 228–234 (cit. on pp. [43](#), [106–108](#), [111](#), [112](#), [117](#)).
- [Don98] E. K. Donald. *The art of computer programming. Sorting and searching*. Second. Vol. 3. Addison Wesley Longman Publishing Co., Inc., 1998, p. 518 (cit. on p. [115](#)).
- [Don+15] C. Dong, C. C. Loy, K. He, and X. Tang. “Image super-resolution using deep convolutional networks.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 38.2 (2015), pp. 295–307 (cit. on p. [82](#)).
- [Don+20] Z. Dong, F. Liang, B. Yang, Y. Xu, Y. Zang, J. Li, Y. Wang, W. Dai, H. Fan, J. Hyypä, et al. “Registration of large-scale terrestrial laser scanner point clouds: A review and benchmark.” In: *ISPRS Journal of Photogrammetry and Remote Sensing* 163 (2020), pp. 327–342 (cit. on pp. [43](#), [105](#)).
- [Dos+20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.” In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020 (cit. on pp. [30](#), [31](#)).
- [DMM22] H. Dröge, T. Möllenhoff, and M. Möller. “Non-smooth energy dissipating networks.” In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE. 2022, pp. 3281–3285 (cit. on p. [87](#)).
- [Du+20] R. Du, E. Turner, M. Dzitsiuk, L. Prasso, I. Duarte, J. Dourgarian, J. Afonso, J. Pascoal, J. Gladstone, N. Cruces, et al. “DepthLab: Real-time 3D interaction with depth maps for mobile augmented reality.” In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 2020, pp. 829–843 (cit. on p. [89](#)).
- [DHS11] J. Duchi, E. Hazan, and Y. Singer. “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of Machine Learning Research* 12.7 (2011) (cit. on p. [35](#)).
- [EF15] D. Eigen and R. Fergus. “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2015, pp. 2650–2658 (cit. on pp. [90](#), [92](#)).

- [EMV21] EMVA. *Standard for Characterization of Image Sensors and Cameras*. en. Standard 1288 Release 4.0 Linear. European Machine Vision Association, 2021. URL: <https://www.emva.org/standards-technology/emva-1288/> (cit. on p. 17).
- [Eve01] C. Everitt. *Interactive order-independent transparency*. Tech. rep. Nvidia, 2001 (cit. on pp. 108, 111).
- [Fer+13] D. Ferstl, C. Reinbacher, R. Ranftl, M. R  ther, and H. Bischof. "Image guided depth upsampling using anisotropic total generalized variation." In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2013, pp. 993–1000 (cit. on p. 91).
- [Fey+18] M. Fey, J. E. Lenssen, F. Weichert, and H. M  ller. "SplineCNN: Fast Geometric Deep Learning With Continuous B-Spline Kernels." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018 (cit. on p. 50).
- [Fla73] H. Flanders. "Differentiation under the integral sign." In: *The American Mathematical Monthly* 80.6 (1973), pp. 615–627 (cit. on p. 56).
- [FP12] D. A. Forsyth and J. Ponce. *Computer Vision - A Modern Approach*. Pearson Education, 2012 (cit. on p. 7).
- [Fu+12] J. Fu, S. Wang, Y. Lu, S. Li, and W. Zeng. "Kinect-like depth denoising." In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2012, pp. 512–515 (cit. on p. 91).
- [Fu+20] Y. Fu, T. Zhang, Y. Zheng, D. Zhang, and H. Huang. "Joint camera spectral response selection and hyperspectral image recovery." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 44.1 (2020), pp. 256–272 (cit. on p. 70).
- [Gha+16] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand. "Deep joint demosaicking and denoising." In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), pp. 1–12 (cit. on p. 82).
- [GB10] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256 (cit. on p. 34).
- [GYBo4] S. B. Gokturk, H. Yalcin, and C. Bamji. "A time-of-flight depth sensor-system description, issues and solutions." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE. 2004, pp. 35–35 (cit. on p. 22).

- [Gos+13] P. Goswami, F. Erol, R. Mukhi, R. Pajarola, and E. Gobbetti. “An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees.” In: *The Visual Computer* 29.1 (2013), pp. 69–83 (cit. on p. 108).
- [Gou+21] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree. “Regularisation of neural networks by enforcing lipschitz continuity.” In: *Machine Learning* 110.2 (2021), pp. 393–416 (cit. on pp. 43, 83).
- [GWo8] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008 (cit. on pp. 36, 37).
- [Guo+18] C. Guo, C. Li, J. Guo, R. Cong, H. Fu, and P. Han. “Hierarchical features driven residual learning for depth map super-resolution.” In: *IEEE Transactions on Image Processing* 28.5 (2018), pp. 2545–2557 (cit. on p. 90).
- [Guo+22] M.-H. Guo, C.-Z. Lu, Q. Hou, Z. Liu, M.-M. Cheng, and S.-m. Hu. “SegNeXt: Rethinking Convolutional Attention Design for Semantic Segmentation.” In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 1140–1156 (cit. on pp. 31, 64).
- [Han+14] A. Handa, T. Whelan, J. McDonald, and A. J. Davison. “A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM.” In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1524–1531 (cit. on p. 118).
- [HKD13] S. Hawe, M. Kleinsteuber, and K. Diepold. “Analysis operator learning and its application to image reconstruction.” In: *IEEE Transactions on Image Processing* 22.6 (2013), pp. 2138–2150 (cit. on p. 42).
- [He+15] K. He, X. Zhang, S. Ren, and J. Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1026–1034 (cit. on p. 34).
- [He+16] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778 (cit. on pp. 30, 66, 98).

- [HK94] G. E. Healey and R. Kondepudy. "Radiometric CCD camera calibration and noise estimation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 16.3 (1994), pp. 267–276 (cit. on pp. 16, 19, 20).
- [Hec17] E. Hecht. *Optics*. 5th ed. Pearson Education, 2017 (cit. on pp. 6, 9, 10, 13, 14).
- [Hec+19] R. Heckel et al. "Deep Decoder: Concise Image Representations from Untrained Non-convolutional Networks." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2019 (cit. on p. 42).
- [Hei+14] F. Heide, M. Steinberger, Y.-T. Tsai, M. Rouf, D. Pająk, D. Reddy, O. Gallo, J. Liu, W. Heidrich, K. Egiazarian, et al. "Flexisp: A flexible camera image processing framework." In: *ACM Transactions on Graphics (TOG)* 33.6 (2014), pp. 1–13 (cit. on pp. 42, 82).
- [HGO18] B. Henz, E. S. L. Gastal, and M. M. Oliveira. "Deep Joint Design of Color Filter Arrays and Demosaicing." In: *Computer Graphics Forum* 37.2 (2018), pp. 389–399 (cit. on pp. 69, 70).
- [HS07] H. Hirschmuller and D. Scharstein. "Evaluation of cost functions for stereo matching." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2007, pp. 1–8 (cit. on pp. 99–102).
- [HLK13] T. Hoegg, D. Lefloch, and A. Kolb. "Real-time motion artifact compensation for PMD-ToF images." In: *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*. Springer. 2013, pp. 273–288 (cit. on p. 22).
- [HK18] A. Holynski and J. Kopf. "Fast depth densification for occlusion-aware augmented reality." In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pp. 1–11 (cit. on p. 89).
- [How+19] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. "Searching for mobilenetv3." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 1314–1324 (cit. on p. 75).
- [Hua+21] Y. Huang, T. Ren, Q. Shen, Y. Fu, and S. You. *HSICityV2: Urban Scene Understanding via Hyperspectral Images*. July 2021. URL: <https://doi.org/10.5281/zenodo.7013301> (cit. on p. 74).
- [HUB64] P. HUBER. "Robust estimation of a location parameter." In: *Ann. Math. Statist.* 35 (1964), pp. 73–101 (cit. on p. 32).

- [Huh+08] B. Huhle, T. Schairer, P. Jenke, and W. Straßer. “Robust non-local denoising of colored depth data.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE. 2008, pp. 1–7 (cit. on p. 91).
- [Hui20] R. Hui. “Chapter 4 - Photodetectors.” In: *Introduction to Fiber-Optic Communications*. Ed. by R. Hui. Academic Press, 2020, pp. 125–154 (cit. on p. 16).
- [HLT16] T.-W. Hui, C. C. Loy, and X. Tang. “Depth map super-resolution by deep multi-scale guidance.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2016, pp. 353–369 (cit. on p. 90).
- [IS15] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. pmlr. 2015, pp. 448–456 (cit. on p. 28).
- [Itu+23] X. Iturbe, N. Abderrahmane, J. Abella, S. Alcaide, E. Beyne, H.-P. Charles, C. Charpin-Nicolle, L. Chittka, A. Dávila, A. Erdmann, et al. “NimbleAI: Towards Neuro-morphic Sensing-Processing 3D-integrated Chips.” In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2023, pp. 1–6 (cit. on p. 48).
- [Jin+22] C. Jin, R. Tanno, T. Mertzaniidou, E. Panagiotaki, and D. C. Alexander. “Learning to Downsample for Segmentation of Ultra-High Resolution Images.” In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2022 (cit. on pp. 49, 50).
- [JAF16] J. Johnson, A. Alahi, and L. Fei-Fei. “Perceptual losses for real-time style transfer and super-resolution.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2016, pp. 694–711 (cit. on pp. 32, 33).
- [JC99] N. P. Jouppi and C.-F. Chang. “Z³: an economical hardware technique for high-quality antialiasing and transparency.” In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*. 1999, pp. 85–93 (cit. on p. 109).
- [KMY17] E. Kang, J. Min, and J. C. Ye. “A deep convolutional neural network using directional wavelets for low-dose X-ray CT reconstruction.” In: *Medical physics* 44.10 (2017), e360–e375 (cit. on p. 82).

- [Kel+13] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. “Real-time 3d reconstruction in dynamic scenes using point-based fusion.” In: *Proceedings of the International Conference on 3D Vision (3DV)*. IEEE. 2013, pp. 1–8 (cit. on pp. [43](#), [44](#), [89](#), [105](#)).
- [KPH21] B. Kim, J. Ponce, and B. Ham. “Deformable kernel networks for joint image filtering.” In: *International Journal of Computer Vision (IJCV)* 129.2 (2021), pp. 579–600 (cit. on pp. [90](#), [91](#), [98](#), [100](#)).
- [KK21] D. Kim and H. Kye. “Z-Thickness Blending: Effective Fragment Merging for Multi-Fragment Rendering.” In: *Computer Graphics Forum* 40.7 (2021), pp. 149–160 (cit. on p. [109](#)).
- [Kin13] D. P. Kingma. *Auto-encoding variational bayes*. 2013. arXiv: [1312.6114 \[stats.ML\]](#) (cit. on p. [73](#)).
- [KB15] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization.” In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015 (cit. on pp. [35](#), [61](#)).
- [Kir10] R. A. Kirsch. “Precision and accuracy in scientific imaging.” In: *Journal of Research of the National Institute of Standards and Technology* 115.3 (2010), p. 195 (cit. on p. [49](#)).
- [Kiv+22] P. E. Kivi, M. J. Mäkitalo, J. Žádník, J. Ikkala, V. K. M. Vadakital, and P. O. Jääskeläinen. “Real-time rendering of point clouds with photorealistic effects: a survey.” In: *IEEE Access* 10 (2022), pp. 13151–13173 (cit. on p. [108](#)).
- [Kli+22] T. Klinghoffer, S. Somasundaram, K. Tiwary, and R. Raskar. “Physics vs. learned priors: Rethinking camera and algorithm design for task-specific imaging.” In: *Proceedings of the IEEE International Conference on Computational Photography (ICCP)*. IEEE. 2022, pp. 1–12 (cit. on p. [48](#)).
- [KN25] J. Klotz and S. K. Nayar. “Minimalist Vision with Freeform Pixels.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2025, pp. 329–346 (cit. on p. [69](#)).
- [Kob+17] E. Kobler, T. Klatzer, K. Hammernik, and T. Pock. “Variational networks: connecting variational methods and deep learning.” In: *Proceedings off the DAGM German Conference on Pattern Recognition (GCPR)*. Springer. 2017, pp. 281–293 (cit. on p. [42](#)).
- [Kol+10] A. Kolb, E. Barth, R. Koch, and R. Larsen. “Time-of-flight cameras in computer graphics.” In: *Computer Graphics Forum* 29.1 (2010), pp. 141–159 (cit. on pp. [21](#), [22](#)).

- [Lai+17] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang. “Deep laplacian pyramid networks for fast and accurate super-resolution.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 624–632 (cit. on pp. 91, 98).
- [LHK15] M. Lambers, S. Hoberg, and A. Kolb. “Simulation of time-of-flight sensors for evaluation of chip layout variants.” In: *IEEE Sensors Journal* 15.7 (2015), pp. 4019–4026 (cit. on p. 118).
- [Lan00] R. Lange. “3D time-of-flight distance measurement with custom solid-state image sensors in CMOS/CCD-technology.” PhD thesis. Universität Siegen, 2000 (cit. on p. 21).
- [Lee+15] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. “Deeply-supervised nets.” In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. Pmlr. 2015, pp. 562–570 (cit. on p. 74).
- [LPW11] W. Lee, N. Park, and W. Woo. “Depth-assisted real-time 3D object detection for augmented reality.” In: *Proceedings of the International Conference on Artificial Reality and Telexistence (ICAT)*. Vol. 11. 2. 2011, pp. 126–132 (cit. on p. 89).
- [LHL13] S. Lefebvre, S. Hornus, and A. Lasram. “HA-Buffer: Coherent Hashing for single-pass A-buffer.” PhD thesis. INRIA, 2013 (cit. on p. 109).
- [Lef+17] D. Lefloch, M. Kluge, H. Sarbolandi, T. Weyrich, and A. Kolb. “Comprehensive use of curvature for robust and accurate online surface reconstruction.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 39.12 (2017), pp. 2349–2365 (cit. on p. 105).
- [LWK15] D. Lefloch, T. Weyrich, and A. Kolb. “Anisotropic point-based fusion.” In: *2015 18th International Conference on Information Fusion (Fusion)*. IEEE. 2015, pp. 2121–2128 (cit. on pp. 43, 44, 105).
- [LW85] M. Levoy and T. Whitted. *The use of points as a display primitive*. Tech. rep. University of North Carolina, Department of Computer Science Chapel Hill, NC ..., 1985 (cit. on pp. 105, 107).
- [LGW19] J. Li, W. Gao, and Y. Wu. “High-quality 3d reconstruction with depth super-resolution and completion.” In: *IEEE Access* 7 (2019), pp. 19370–19381 (cit. on pp. 90, 91, 98, 100).

- [LDV23] K. Li, D. Dai, and L. Van Gool. “Jointly learning band selection and filter array design for hyperspectral imaging.” In: *Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV)*. 2023, pp. 6384–6394 (cit. on p. 70).
- [Li+20] L. Li, J. Pan, W.-S. Lai, C. Gao, N. Sang, and M.-H. Yang. “Dynamic scene deblurring by depth guided model.” In: *IEEE Transactions on Image Processing* 29 (2020), pp. 5273–5288 (cit. on pp. 90, 91).
- [Liu+09] F. Liu, M.-C. Huang, X.-H. Liu, and E.-H. Wu. “Efficient depth peeling via bucket sort.” In: *Proceedings of the Conference on High Performance Graphics*. 2009, pp. 51–57 (cit. on p. 108).
- [Liu+10] F. Liu, M.-C. Huang, X.-H. Liu, and E.-H. Wu. “Freepipe: a programmable parallel rendering architecture for efficient multi-fragment effects.” In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. 2010, pp. 75–82 (cit. on p. 109).
- [Liu+18a] R. Liu, S. Cheng, L. Ma, X. Fan, Z. Luo, et al. “A bridging framework for model optimization and deep propagation.” In: *Advances in Neural Information Processing Systems* 31 (2018) (cit. on p. 42).
- [Liu+18b] R. Liu, L. Ma, Y. Wang, and L. Zhang. “Learning converged propagations with deep prior ensemble for image enhancement.” In: *IEEE Transactions on Image Processing* 28.3 (2018), pp. 1528–1543 (cit. on p. 42).
- [Liu+21] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2021 (cit. on p. 31).
- [Liu+22] Z. Liu et al. “Swin Transformer V2: Scaling Up Capacity and Resolution.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 (cit. on p. 31).
- [Liu+15] Z. Liu, P. Luo, X. Wang, and X. Tang. “Deep Learning Face Attributes in the Wild.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. Dec. 2015 (cit. on p. 66).
- [LH16] I. Loshchilov and F. Hutter. *Sgdr: Stochastic gradient descent with warm restarts*. 2016. arXiv: 1608.03983 [cs.LG] (cit. on p. 36).
- [LH17] I. Loshchilov and F. Hutter. *Decoupled weight decay regularization*. 2017. arXiv: 1711.05101 [cs.LG] (cit. on p. 36).

- [Mar+19] D. Marin, Z. He, P. Vajda, P. Chatterjee, S. Tsai, F. Yang, and Y. Boykov. “Efficient segmentation: Learning down-sampling near semantic boundaries.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2019, pp. 2131–2141 (cit. on pp. 49, 50).
- [MW21] J. Martel and G. Wetzstein. “Computational imaging with vision sensors embedding in-pixel processing.” In: *IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2021, pp. 35–3 (cit. on p. 48).
- [McG17] M. McGuire. *Computer Graphics Archive*. July 2017. URL: <https://casual-effects.com/data> (cit. on p. 118).
- [Mei+17] T. Meinhardt, M. Moller, C. Hazirbas, and D. Cremers. “Learning proximal operators: Using denoising networks for regularizing inverse imaging problems.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2017, pp. 1781–1790 (cit. on p. 43).
- [MBC16] R. Mekuria, K. Blom, and P. Cesar. “Design, implementation, and evaluation of a point cloud codec for tele-immersive video.” In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.4 (2016), pp. 828–842 (cit. on p. 43).
- [Men+22] L. Mennel, A. J. Molina-Mendoza, M. Paur, D. K. Polyushkin, D. Kwak, M. Giparakis, M. Beiser, A. M. Andrews, and T. Mueller. “A photosensor employing data-driven binning for ultrafast image recognition.” In: *Scientific Reports* 12.1 (2022), p. 14441 (cit. on p. 51).
- [Met+22] G. Metzger, R. Hanocka, R. Giryes, N. J. Mitra, and D. Cohen-Or. “Z2p: Instant visualization of point clouds.” In: *Computer Graphics Forum* 41.2 (2022), pp. 461–471 (cit. on p. 108).
- [Met+20] C. A. Metzler, H. Ikoma, Y. Peng, and G. Wetzstein. “Deep optics for single-shot high-dynamic-range imaging.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 1375–1385 (cit. on pp. 47, 50, 69).
- [Mic+20] A. Michel, W. Gross, F. Schenkel, and W. Middelmann. “Hyperspectral band selection within a deep reinforcement learning framework.” In: *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*. IEEE. 2020, pp. 52–55 (cit. on p. 70).
- [Miy+18] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. “Spectral Normalization for Generative Adversarial Networks.” In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018 (cit. on p. 83).

- [MMS20] MMSegmentation Contributors. *OpenMMLab Semantic Segmentation Toolbox and Benchmark*. July 2020. URL: <https://github.com/open-mmlab/mmdetection> (cit. on p. 75).
- [MC18] M. Moeller and D. Cremers. “Image denoising—old and new.” In: *Denoising of Photographic Images and Video: Fundamentals, Open Challenges and New Trends*. Springer, 2018, pp. 63–91 (cit. on p. 82).
- [MMC19] M. Moeller, T. Mollenhoff, and D. Cremers. “Controlling neural networks via energy dissipation.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2019, pp. 3256–3265 (cit. on pp. 84–87).
- [Mos+20] A. Mosleh, A. Sharma, E. Onzon, F. Mannan, N. Robidoux, and F. Heide. “Hardware-in-the-loop end-to-end optimization of camera image processing pipelines.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 7529–7538 (cit. on pp. 47, 49, 50, 69).
- [Mou+21] L. Mou, S. Saha, Y. Hua, F. Bovolo, L. Bruzzone, and X. X. Zhu. “Deep reinforcement learning for band selection in hyperspectral image classification.” In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2021), pp. 1–14 (cit. on p. 70).
- [NIL12] C. V. Nguyen, S. Izadi, and D. Lovell. “Modeling kinect sensor noise for improved 3d reconstruction and tracking.” In: *2012 second international conference on 3D imaging, modeling, processing, visualization & transmission*. IEEE. 2012, pp. 524–530 (cit. on p. 22).
- [NMW22] C. M. Nguyen, J. N. Martel, and G. Wetzstein. “Learning spatially varying pixel exposures for motion deblurring.” In: *Proceedings of the IEEE International Conference on Computational Photography (ICCP)*. IEEE. 2022, pp. 1–11 (cit. on p. 48).
- [Nie+18] S. Nie, L. Gu, Y. Zheng, A. Lam, N. Ono, and I. Sato. “Deeply learned filter response functions for hyperspectral reconstruction.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 4767–4776 (cit. on pp. 70, 72, 73).
- [Nim+20] M. Nimier-David, S. Speierer, B. Ruiz, and W. Jakob. “Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering.” In: *ACM Transactions on Graphics (TOG)* 39.4 (July 2020) (cit. on p. 19).

- [PB+14] N. Parikh, S. Boyd, et al. “Proximal algorithms.” In: *Foundations and trends® in Optimization* 1.3 (2014), pp. 127–239 (cit. on pp. 39, 41, 42).
- [Pau+03] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. “Shape modeling with point-sampled geometry.” In: *ACM Transactions on Graphics (TOG)* 22.3 (2003), pp. 641–650 (cit. on pp. 107, 121).
- [PJH23] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. 3rd ed. The MIT Press, 2023, p. 1280 (cit. on pp. 14, 15, 19).
- [PGA11] R. Pintus, E. Gobbetti, and M. Agus. “Real-time rendering of massive unstructured raw point clouds using screen-space operators.” In: *Proceedings of the 12th International conference on Virtual Reality, Archaeology and Cultural Heritage*. 2011, pp. 105–112 (cit. on p. 108).
- [PCS+15] F. Pomerleau, F. Colas, R. Siegwart, et al. “A review of point cloud registration algorithms for mobile robotics.” In: *Foundations and Trends® in Robotics* 4.1 (2015), pp. 1–104 (cit. on pp. 43, 105).
- [PC82] M. Potmesil and I. Chakravarty. “Synthetic Image Generation with a Lens and Aperture Camera Model.” In: *ACM Transactions on Graphics (TOG)* 1.2 (1982), pp. 85–108 (cit. on p. 10).
- [PLC19] R. P. Poudel, S. Liwicki, and R. Cipolla. *Fast-scnn: Fast semantic segmentation network*. 2019. arXiv: 1902.04502 [cs.CV] (cit. on p. 75).
- [P JW12] R. Preiner, S. Jeschke, and M. Wimmer. “Auto Splats: Dynamic Point Cloud Visualization on the GPU.” In: *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*. Ed. by H. Childs and T. Kuhlen. Eurographics Association 2012. Cagliari, May 2012, pp. 139–148 (cit. on p. 117).
- [Qi+18] X. Qi, R. Liao, Z. Liu, R. Urtasun, and J. Jia. “GeoNet: Geometric neural network for joint depth and surface normal estimation.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 283–291 (cit. on pp. 90, 92, 97).
- [Qi+20] X. Qi, Z. Liu, R. Liao, P. H. Torr, R. Urtasun, and J. Jia. “GeoNet++: Iterative geometric neural network with edge-aware refinement for joint depth and surface normal estimation.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 44.2 (2020), pp. 969–984 (cit. on pp. 90, 92–96, 103).

- [Ria+22] R. Riad, O. Teboul, D. Grangier, and N. Zeghidour. "Learning Strides in Convolutional Neural Networks." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2022 (cit. on p. 50).
- [RRB16] G. Riegler, M. Rüther, and H. Bischof. "ATGV-Net: Accurate depth super-resolution." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2016, pp. 268–284 (cit. on pp. 90, 98, 99).
- [RW98] R. T. Rockafellar and R. J. Wets. *Variational analysis*. Springer, 1998 (cit. on pp. 39, 40).
- [REM17] Y. Romano, M. Elad, and P. Milanfar. "The little engine that could: Regularization by denoising (RED)." In: *Journal on Imaging Sciences* 10.4 (2017), pp. 1804–1844 (cit. on p. 82).
- [RFB15] O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation." In: *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. Springer. 2015, pp. 234–241 (cit. on p. 49).
- [RB09] S. Roth and M. J. Black. "Fields of experts." In: *International Journal of Computer Vision (IJCV)* 82.2 (2009), pp. 205–229 (cit. on p. 42).
- [ROF92] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear total variation based noise removal algorithms." In: *Physica D: nonlinear phenomena* 60.1-4 (1992), pp. 259–268 (cit. on p. 39).
- [RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 35).
- [RL00] S. Rusinkiewicz and M. Levoy. "QSplat: A multiresolution point rendering system for large meshes." In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 343–352 (cit. on p. 105).
- [Ryu+19] E. Ryu, J. Liu, S. Wang, X. Chen, Z. Wang, and W. Yin. "Plug-and-play methods provably converge with properly trained denoisers." In: *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR. 2019, pp. 5546–5557 (cit. on pp. 43, 82).
- [RB16] E. K. Ryu and S. Boyd. "Primer on monotone operator methods." In: *Appl. comput. math* 15.1 (2016), pp. 3–43 (cit. on p. 41).
- [Sch11] M. Schmidt. "Analysis, modeling and dynamic optimization of 3d time-of-flight imaging systems." PhD thesis. Universität Heidelberg, 2011 (cit. on pp. 21, 22).

- [SR14] U. Schmidt and S. Roth. “Shrinkage fields for effective image restoration.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 2774–2781 (cit. on p. 42).
- [SGL18] H. Sedghi, V. Gupta, and P. M. Long. *The singular values of convolutional layers*. 2018. arXiv: 1805.10408 [cs.LG] (cit. on pp. 43, 83).
- [See+18] P. Seemann, G. Palma, M. Dellepiane, P. Cignoni, and M. Goesele. “Soft transparency for point cloud rendering.” In: *Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations*. 2018, pp. 95–106 (cit. on pp. 107, 108).
- [SC13] J. Shen and S.-C. S. Cheung. “Layer depth denoising and completion for structured-light RGB-D cameras.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 1187–1194 (cit. on p. 91).
- [Sil+12] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. “Indoor segmentation and support inference from RGBD images.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2012, pp. 746–760 (cit. on pp. 96–98, 100–102).
- [14] *Silicon Photodiode*. FDS10x10. Thor Labs. Nov. 2014 (cit. on p. 17).
- [Som+24] H. Sommerhoff, S. Agnihotri, M. Saleh, M. Moeller, M. Keuper, B. Choubey, and A. Kolb. “Task-driven sensor layouts - Joint optimization of pixel layout and network parameters.” In: *Proceedings of the IEEE International Conference on Computational Photography (ICCP)*. 2024, pp. 1–10 (cit. on pp. 2, 47, 69).
- [SK22] H. Sommerhoff and A. Kolb. “A generic framework for depth reconstruction enhancement.” In: *Journal of Imaging* 8.5 (2022), p. 138 (cit. on pp. 3, 89).
- [SK24] H. Sommerhoff and A. Kolb. “Hashed, binned A-buffer for real-time outlier removal and rendering of noisy point clouds.” In: *The Visual Computer* 40.3 (2024), pp. 1825–1838 (cit. on pp. 3, 105).
- [SKM19] H. Sommerhoff, A. Kolb, and M. Moeller. “Energy dissipation with plug-and-play priors.” In: *NeurIPS Workshop on Solving Inverse Problems with Deep Networks*. 2019 (cit. on pp. 2, 81).

- [Sre+16] S. Sreehari, S. V. Venkatakrishnan, B. Wohlberg, G. T. Buzard, L. F. Drummy, J. P. Simmons, and C. A. Bouman. “Plug-and-play priors for bright field electron tomography and sparse interpolation.” In: *IEEE Transactions on Computational Imaging* 2.4 (2016), pp. 408–423 (cit. on p. 82).
- [Ste+19a] H. Steiner, H. Sommerhoff, D. Bulczak, N. Jung, M. Lambers, and A. Kolb. “Fast motion estimation for field sequential imaging: Survey and benchmark.” In: *Image and Vision Computing* 89 (2019), pp. 170–182 (cit. on pp. 22, 89).
- [Ste+19b] V. Sterzentsenko, L. Saroglou, A. Chatzitofis, S. Thermos, N. Zioulis, A. Doumanoglou, D. Zarpalas, and P. Daras. “Self-supervised deep depth denoising.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2019, pp. 1242–1251 (cit. on p. 90).
- [Sun+20] Q. Sun, E. Tseng, Q. Fu, W. Heidrich, and F. Heide. “Learning rank-1 diffractive optics for single-shot high dynamic range imaging.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 1386–1396 (cit. on pp. 47, 50, 69).
- [Suo+23] J. Suo, W. Zhang, J. Gong, X. Yuan, D. J. Brady, and Q. Dai. “Computational Imaging and Artificial Intelligence: The Next Revolution of Mobile Vision.” In: *Proceedings of the IEEE* 111.12 (2023), pp. 1607–1639 (cit. on p. 48).
- [Sze22] R. Szeliski. *Computer Vision: Algorithms and Applications*. 2nd ed. Springer, 2022 (cit. on pp. 5, 6, 9, 10).
- [TM21] H. Talebi and P. Milanfar. “Learning to resize images for computer vision tasks.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2021, pp. 497–506 (cit. on p. 49).
- [TCZ21] J. Tang, X. Chen, and G. Zeng. “Joint implicit image function for guided depth super-resolution.” In: *Proceedings of the ACM International Conference on Multimedia*. 2021, pp. 4390–4399 (cit. on pp. 90, 91, 98, 100).
- [Tao+18] X. Tao, H. Gao, X. Shen, J. Wang, and J. Jia. “Scale-recurrent network for deep image deblurring.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8174–8182 (cit. on p. 82).
- [Tia+20] C. Tian, Y. Xu, Z. Li, W. Zuo, L. Fei, and H. Liu. “Attention-guided CNN for image denoising.” In: *Neural Networks* 124 (2020), pp. 117–129 (cit. on p. 97).

- [Tou+16] S. Tourani, S. Mittal, A. Nagariya, V. Chari, and M. Krishna. “Rolling shutter and motion blur removal for depth cameras.” In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 5098–5105 (cit. on pp. 90, 91).
- [Tse+21] E. Tseng, A. Mosleh, F. Mannan, K. St-Arnaud, A. Sharma, Y. Peng, A. Braun, D. Nowrouzezahrai, J.-F. Lalonde, and F. Heide. “Differentiable Compound Optics and Processing Pipeline Optimization for End-to-end Camera Design.” In: *ACM Transactions on Graphics (TOG)* 40.2 (2021), pp. 1–19 (cit. on pp. 47, 50, 69).
- [Uch+20] T. Uchida, K. Hasegawa, L. Li, M. Adachi, H. Yamaguchi, F. I. Thufail, S. Riyanto, A. Okamoto, and S. Tanaka. “Noise-robust transparent visualization of large-scale point clouds acquired by laser scanning.” In: *ISPRS Journal of Photogrammetry and Remote Sensing* 161 (2020), pp. 124–134 (cit. on pp. 107, 108).
- [UVL16] D. Ulyanov, A. Vedaldi, and V. Lempitsky. *Instance normalization: The missing ingredient for fast stylization*. 2016. arXiv: 1607.08022 [cs.CV] (cit. on p. 29).
- [UVL18] D. Ulyanov, A. Vedaldi, and V. Lempitsky. “Deep image prior.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 9446–9454 (cit. on p. 42).
- [VVP20] A.-A. Vasilakis, K. Vardis, and G. Papaioannou. “A survey of multifragment rendering.” In: *Computer Graphics Forum* 39.2 (2020), pp. 623–642 (cit. on pp. 106, 108).
- [Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need.” In: *Advances in Neural Information Processing Systems* 30 (2017) (cit. on p. 31).
- [VBW13] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg. “Plug-and-play priors for model based reconstruction.” In: *Proceedings of the IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE. 2013, pp. 945–948 (cit. on pp. 42, 82).
- [VS18] A. Virmaux and K. Scaman. “Lipschitz regularity of deep neural networks: analysis and efficient estimation.” In: *Advances in Neural Information Processing Systems* 31 (2018) (cit. on p. 83).
- [Wan+16] P. Wang, X. Shen, B. Russell, S. Cohen, B. Price, and A. L. Yuille. “Surge: Surface regularized geometry estimation from a single image.” In: *Advances in Neural Information Processing Systems* 29 (2016) (cit. on pp. 90, 92).

- [Wan+04] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image quality assessment: from error visibility to structural similarity." In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612 (cit. on p. 99).
- [Wen95] H. Wendland. "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree." In: *Advances in Computational Mathematics* 4.1 (1995), pp. 389–396 (cit. on p. 110).
- [Wes+09] A. H. Westra, J. W. Heemskerk, M. A. Korevaar, A. J. Theuwissen, R. Kreuger, K. M. Ligtoet, and F. J. Beekman. "On-chip pixel binning in photon-counting EMCCD-based gamma camera: a powerful tool for noise reduction." In: *IEEE Transactions on Nuclear Science* 56.5 (2009), pp. 2559–2565 (cit. on p. 51).
- [Whe+15] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. "ElasticFusion: Dense SLAM without a pose graph." In: *Robotics: Science and Systems*. Vol. 11. 3. Rome. 2015 (cit. on pp. 43, 44, 105).
- [Wil+17] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. "The marginal value of adaptive gradient methods in machine learning." In: *Advances in Neural Information Processing Systems* 30 (2017) (cit. on p. 36).
- [Xu+15] B. Xu, N. Wang, T. Chen, and M. Li. *Empirical evaluation of rectified activations in convolutional network*. 2015. arXiv: 1505.00853 [cs.LG] (cit. on p. 26).
- [Xu+18] D. Xu, W. Ouyang, X. Wang, and N. Sebe. "PAD-Net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 675–684 (cit. on pp. 90, 92).
- [Xu+14] L. Xu, J. S. Ren, C. Liu, and J. Jia. "Deep convolutional neural network for image deconvolution." In: *Advances in Neural Information Processing Systems* 27 (2014) (cit. on p. 82).
- [Yan+18] S. Yan, C. Wu, L. Wang, F. Xu, L. An, K. Guo, and Y. Liu. "DDRNet: Depth map denoising and refinement for consumer depth cameras using cascaded cnns." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 151–167 (cit. on p. 90).
- [Yan+17] G. Yang, S. Yu, H. Dong, G. Slabaugh, P. L. Dragotti, X. Ye, F. Liu, S. Arridge, J. Keegan, Y. Guo, et al. "DAGAN: deep de-aliasing generative adversarial networks for fast

- compressed sensing MRI reconstruction." In: *IEEE Transactions on Medical Imaging* 37.6 (2017), pp. 1310–1321 (cit. on p. 82).
- [Yan+10] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz. "Real-time concurrent linked list construction on the GPU." In: *Computer Graphics Forum* 29.4 (2010), pp. 1297–1304 (cit. on p. 109).
- [YIP15] Y. Yoo, J. Im, and J. Paik. "Low-light image enhancement using adaptive digital pixel binning." In: *Sensors* 15.7 (2015), pp. 14917–14931 (cit. on p. 51).
- [You+20] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger. "Pseudo-LiDAR++: Accurate Depth for 3D Object Detection in Autonomous Driving." In: *International Conference on Learning Representations (ICLR)*. 2020 (cit. on p. 89).
- [YK16] F. Yu and V. Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016 (cit. on p. 27).
- [YF21] L. Yu and G. Fan. "DrsNet: Dual-resolution Semantic Segmentation with Rare Class-Oriented Superpixel Prior." In: *Multimedia Tools and Applications* 80 (Jan. 2021), pp. 1–20 (cit. on p. 50).
- [Zha+18] J. Zhang, J. Jia, A. Sheng, and K. Hirakawa. "Pixel binning for high dynamic range color image sensor using square sampling lattice." In: *IEEE Transactions on Image Processing* 27.5 (2018), pp. 2229–2241 (cit. on p. 50).
- [Zha+17a] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising." In: *IEEE Transactions on Image Processing* 26.7 (2017), pp. 3142–3155 (cit. on pp. 82–85, 97, 98).
- [Zha+25] Q. Zhang, H. Sommerhoff, M. Möller, A. Kolb, and B. Choubey. "Curvilinear Layout for CMOS Image Sensors: Task-Specific Imaging." In: *International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. 2025 (cit. on pp. 48, 55, 130).
- [Zha00] Z. Zhang. "A flexible new technique for camera calibration." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 22.11 (2000), pp. 1330–1334 (cit. on p. 13).

- [Zha+17b] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. “Pyramid scene parsing network.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2881–2890 (cit. on pp. [49](#), [64](#), [65](#)).
- [Zha+19] L. Zhao, H. Bai, J. Liang, B. Zeng, A. Wang, and Y. Zhao. “Simultaneous color-depth super-resolution with conditional generative adversarial networks.” In: *Pattern Recognition* 88 (2019), pp. 356–369 (cit. on p. [91](#)).
- [Zhe+15] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. “Conditional random fields as recurrent neural networks.” In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2015, pp. 1529–1537 (cit. on p. [42](#)).
- [Zho+21] Z. Zhong, X. Liu, J. Jiang, D. Zhao, Z. Chen, and X. Ji. “High-resolution depth maps imaging via attention-based hierarchical multi-modal fusion.” In: *IEEE Transactions on Image Processing* 31 (2021), pp. 648–663 (cit. on p. [92](#)).
- [Zwi+01] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross. “Surface splatting.” In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001, pp. 371–378 (cit. on pp. [45](#), [46](#), [107](#), [109](#)).
- [Zwi+02] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross. “EWA splatting.” In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 8.3 (2002), pp. 223–238 (cit. on p. [46](#)).