The Creation of Virtual Scenes and their Realistic Embedding in the Real Environment

Dissertation

submitted for the award of the title

“Doctor of Natural Sciences”

to the Faculty of Physics, Mathematics, and Computer Science
of Johannes Gutenberg University Mainz
in Mainz

Alexander Sommer

Born in Diez

Mainz, February 26, 2025

Alexander Sommer
The Creation of Virtual Scenes and their Realistic Embedding in the Real Environment

First reviewer: [REMOVED]
Second reviewer: [REMOVED]

Date of oral examination: September 18, 2025

Johannes Gutenberg University Mainz
Institute of Computer Science
Staudingerweg 9
55128 Mainz

Abstract

The creation of virtual worlds has fascinated researchers for several decades. Advances in hardware have made it ever more feasible for the virtual and real worlds to merge seamlessly in the field of mixed reality (MR). This research field combines a wide range of disciplines, such as computer graphics, computer vision, machine learning, and physics. Despite great progress in the last decade, several challenges remain, particularly in achieving real-time performance on devices with limited hardware resources or in providing convincing visual quality with existing methods.

This thesis explores various aspects of MR and presents methods to improve the quality of applications in this domain. It includes a set of techniques for sampling the volume and surface of 3D objects using discrete points. Building on these techniques, a novel method for real-time simulation of granular materials with large particle numbers is introduced. By structuring the simulation in two stages, the method first simulates a small number of particles, including all acting forces. Subsequently, a larger number of particles are added for visualization, partially following the velocity field defined by the first stage. This approach allows for a more immersive experience in virtual reality applications where interactions with materials like sand or gravel are simulated, such as in heavy equipment operator training.

Moreover, this thesis focuses on improving the photometric registration of virtual objects in augmented reality applications. A method for analyzing the real-world lighting situation is presented, using geometric and photometric parameters derived from an RGB camera image with an artificial neural network. Furthermore, a novel method for the efficient visualization of soft cast shadows of virtual objects is introduced. In this approach, shadow textures are encoded within tiny artificial neural networks, which can be queried based on the current lighting situation to produce realistic shadows with minimal computational overhead.

Acknowledgements

[REMOVED]

Nomenclature

Sets

$\mathbb{N}$	The set of natural numbers without zero
$\mathbb{N}_0$	The set of natural numbers including zero
$\mathbb{Z}$	The set of all integers
$\mathbb{R}$	The set of real numbers
$\mathbb{P}^n$	The n-dimensional projective space
$\{i, \ldots, j\}$	A set of integer values from i to j
$[v_1, v_2]$	The range of real values from v_1 to v_2
$\mathbb{S}$	A set of points
$\lvert\mathbb{S}\rvert$	The number of points in $\mathbb{S}$

Symbols

$\boldsymbol{I}$	An image / a matrix
$\boldsymbol{I}(u, v)$	The intensity information at pixel coordinate (u, v) of image $\boldsymbol{I}$
$\boldsymbol{x}$	A vector
$\hat{\boldsymbol{x}}$	A unit vector
$\tilde{\boldsymbol{x}}$	A homogeneous vector

Functions

$\odot$	The element-wise product of two matrices with same dimensions
$\lfloor x \rfloor$	Floor function: The greatest integer less than or equal to x
$\lceil x \rceil$	Ceiling function: The least integer greater than or equal to x
$\lfloor x \rceil$	Rounding function: The integer nearest to x

List of Abbreviations

2D	two-dimensional
3D	three-dimensional
AI	artificial intelligence
AR	augmented reality
BRDF	bidirectional reflectance distribution function
BTF	bidirectional texture function
CFL	Courant-Friedrichs-Lewy
CNN	convolutional neural network
DEM	discrete element method
DenseNet	densely connected convolutional network
DoF	degrees of freedom
ELU	exponential linear unit
EV	exposure value
FCC	face-centered cubic
FLIP	fluid implicit particle
FOV	field of view
HCP	hexagonal close packing
HDM	head-mounted display
HDR	high dynamic range
IBL	image-based lighting
LDR	low dynamic range
ML	machine learning

MLP	multi-layer perceptron
MPM	material point method
MR	mixed reality
MSE	mean squared error
NeRF	neural radiance field
PBD	position-based dynamics
ReLU	rectified linear unit
RMLE	relative mean logarithmic error
RMSE	root-mean-square error
si-RMSE	scale-invariant root-mean-square error
SPH	smoothed particle hydrodynamics
tanh	hyperbolic tangent
VR	virtual reality

Contents

1

Introduction

The world around us, as we experience it through our senses, defines our concept of reality. From a more scientific perspective, this reality is often referred to as the *real environment* in various research disciplines. Over the last 50 years, advances in computer graphics have made it possible to recreate large parts of this real environment synthetically. This results in the creation of virtual scenes or entire virtual worlds, referred to as *virtual environments*. These virtual environments are the foundation of a second, completely digital reality known as virtual reality (VR). The goal of VR is to immerse the user in these virtual environments. This is achieved by simulating different sensory inputs; for example, head tracking [RBG01; KM20] can be used to transfer the user's real head movement to the camera movement in the virtual world.

VR has its origins already back in the 1960s with the pioneering work of Ivan Sutherland [S65]. In the 1980s, the development of VR gained momentum from companies such as VPL Research, founded by Thomas Zimmerman and Jaron Lanier, who also popularized the term *virtual reality* [L17]. However, these early developments were limited to a small group of users, as they were mainly used for highly specialized applications in industry, the military, and academia. It was not until the 2010s that VR became accessible to a broader audience when the Oculus Rift [DDA+14] became the first small, easy-to-use, and, most importantly, affordable VR system for individual users. Since then, VR has reached the mainstream and continues to evolve. It shows growth potential in industries such as gaming [B23], education [MTK+23], and healthcare [KKB+23].

The real and virtual environments form the boundaries of the *reality-virtuality continuum*, conceptualized by Paul Milgram [MTU+95]. In this continuous spectrum—more commonly referred to as *mixed reality (MR)*—the lines between the physical and virtual worlds become increasingly blurred. Augmented reality (AR) is the most prominent example of a hybrid reality within this continuum. AR is positioned closer to the real environment than its lesser-known counterpart—*augmented virtuality*—which incorporates

Figure 1.1: Several virtual pieces of furniture and decor items inserted into the real environment using the *IKEA Kreativ* room planner.

real-world elements into a primarily virtual space.

While the definitions are not entirely rigid, AR typically involves superimposing virtual content onto the real environment. This synthesis aims to enhance the user's perception with additional information. AR is used for educational purposes, such as learning [AJG+23] and tourism [LE20], or to make interactions with the physical world more intuitive, finding applications in maintenance [PER+18], product assembly [EGI+23], and medicine [EVF19]. Visualizations in AR can also assist in project planning, with applications in architecture [YNE23] and construction [DOD+20], as well as in retail and e-commerce [LMT22]. For example, customers can visualize furniture in their homes before purchasing (see Figure 1.1). AR has also gained popularity for pure entertainment purposes, such as gaming [MG22] with titles like Pokémon Go[1].

AR is accessible through a wide range of devices, from handheld smartphones and tablets that display digital content over the camera feed to specialized head-mounted displays (HDMs). HDMs can be classified into see-through and non-see-through displays. Non-see-through HDMs include dedicated VR headsets as well as devices such as Apple's Vision Pro, which use cameras to capture the real environment and display an augmented view on their screens. In contrast, see-through HDMs (e.g., Microsoft's HoloLens) allow the user to perceive the real environment by looking through transparent displays. In

[1]Pokémon Go was released in 2016 by Niantic and is available at: `https://pokemongolive.com/`

these devices—also known as AR glasses—virtual content is projected directly into the user's field of view. The development of HDMs continues [MBW+23] to make them smaller and more suitable for everyday use.

A key factor for the success of an AR application is ensuring that the virtual elements blend seamlessly into the real environment. This requires an accurate understanding of the physical properties of the real environment. For this purpose, most AR devices are equipped with one or more cameras to capture the surroundings. Therefore, it is essential to understand how the two-dimensional (2D) camera image is formed as a projection of the 3D real world. This image formation process can be described by a mathematical model, referred to as *camera geometry* in computer vision (see Section 2.2.1). This model can be used to determine the position and orientation of the camera relative to the real environment, forming the basis for developing a deeper understanding of the environment. For example, *depth estimation* [WWH+22; YKH+24; RSL+24] allows to determine the distance between the device and any visible point in the real environment. *Feature detection* [TM08; LWT+15; XCX+24] identifies distinct points, such as the corners of a room, the edges of objects, or patterns in textured regions. As the AR device moves, these features ensure the virtual content maintains its position relative to the real-world space. Additionally, *plane detection* [NOB+16; LKG+19; XGY+22] identifies flat surfaces within the real environment, such as floors, tables, or walls, which serve as stable and realistic insertion points for virtual objects.

One of the biggest challenges in achieving realistic blending between the real and the virtual worlds is *photometric registration* [KY02; GRS12; AT20]. This process involves adjusting the visual properties of virtual objects, such as brightness, shadows, and reflections, that they integrate visually into the real environment (see Figure 1.2). Improving photometric registration is one of the central research questions of this thesis. An important substep in this process is *lighting estimation* [BH85; LN16; GHS+19] (see Section 5.2), where the lighting conditions in the real environment are measured using the camera image. The lighting situation is described by an appropriate representation, which may include parameters such as light direction, intensity, and the color of the light source.

This lighting representation can then be used in a process called *rendering* [PJH23] to create a 2D projection of the scene-aware illuminated virtual 3D object, which is then superimposed on the real environment. To ensure that the perspective and proportions of the rendered object align with the real environment, it is also important to correctly set the virtual camera angle and the virtual object position for the rendering. The rendering includes not only the plausibly lit object, but also its shadow, which plays a crucial role

Figure 1.2: A 3D-printed model of a squirrel (left) in the real environment next to its photometrically registered virtual counterpart (right) blended over the camera image, including shadows and reflections.

in the seamless integration of the virtual element into the real world. The mathematical foundation of rendering is the *rendering equation* [K86; ICG86] (see Section 2.3.1), which describes how light interacts with different surfaces in the virtual scene. Evaluating this equation is computationally expensive and, in most cases, can only be approximated numerically. Consequently, there are several methods [P75; B77; W79] that simplify the virtual image synthesis process to enable real-time rendering [AHH+18]. However, the calculation of shadows, in particular, remains a complex problem regarding real-time requirements.

Rendering is only one aspect of computer graphics that plays an important role in AR or MR in general. Another important component is animation, which brings virtual objects to life by simulating movement, deformation, and interaction with elements in the real environment. Animation can range from simple transformations, such as moving or rotating a virtual object, to more complex simulations, such as the behavior of fluids [B15] or granular materials [BYM05; KGP+16]. Physically-based animations [HK20] simulate physical phenomena such as gravity, friction, and collisions to ensure that virtual objects behave realistically within the virtual environment or when interacting with the real environment. Particularly challenging is the real-time simulation of virtual objects composed of many smaller elements such as granular materials, which places high computational demands on the limited resources of MR systems.

The other key research question of this thesis is how to accelerate time-critical but essential operations for MR applications, such as shadow calculation and the simulation of complex systems, by leveraging efficient approximations inspired by advances in other research fields.

Overall, this thesis integrates elements from various disciplines, including mixed reality, computer vision, and computer graphics, as well as areas of machine learning and physics. A more detailed classification within the specific disciplines is given in the respective chapters.

1.1 | Publications

The research leading to this thesis started in early 2020 and extended into the summer of 2023. It was conducted within the Computer Vision and Mixed Reality Group at the RheinMain University of Applied Sciences in Wiesbaden in cooperation with the Computational Geometry Group at the Johannes Gutenberg University in Mainz. The methods developed in this research aim to improve MR applications, while each contribution also represents a novel advancement in its specific field.

Initially, we focused on collecting different sampling methods for the volumes and surfaces of 3D objects. This effort included deterministic methods to achieve dense sampling and stochastic techniques to generate samples that meet specific randomness criteria. The result was a collection of methods made available to the research community as a library for existing projects and a stand-alone tool. This work led to the publication *LEAVEN - Lightweight Surface and Volume Mesh Sampling Application for Particle-based Simulations* [SS21]. Sampling plays an important role in many areas of computer science, and the methods collected in this research phase have been used in subsequent research phases.

In the next phase, the focus shifted to physically-based animation, specifically the simulation of granular materials. Granular materials are composed of many small elements, making their simulation typically impractical for real-time performance. We developed a novel approach that divides the simulation into two separate processes: a first, more precise simulation of a small number of elements and a second that builds on the results of the first to simulate a large number of elements efficiently. The sampling methods from the earlier phase also played a role here. Our results, published as *Interactive High-Resolution Simulation of Granular Material* [SSS22], show significant performance

improvements over existing state-of-the-art methods. Our method can be used in MR applications such as heavy equipment operator training, to create a more immersive experience through realistic granular material simulation.

The final phase of the research focused on improving photometric registration in AR applications. We proposed a lighting representation and trained a neural network to estimate the parameters of this representation using only a single RGB camera image from an AR device. Our approach achieves accuracy comparable to other state-of-the-art methods, while using fewer parameters. This parameterization of the lighting situation served as the foundation for redefining the generation of realistic cast shadows. We developed a novel approach that trains tiny neural networks to generate object-specific shadow textures based on the estimated lighting parameters. These networks are small enough to run efficiently in real time on AR devices with minimal memory consumption. The results of this research phase have been published as *Real-time Light Estimation and Neural Soft Shadows for AR Indoor Scenarios* [SSS23].

1.2 | Thesis Outline

This thesis consists of seven chapters, including this introduction. The introduction in Chapter 1 aims to place the research results presented in later chapters in a broader thematic context. Therefore, the field of mixed reality, including its subfields augmented reality and virtual reality, is introduced and connected to the fields of computer vision and computer graphics.

Chapter 2 establishes a consistent notation for the rest of the thesis. It also provides a brief overview of basic concepts that are assumed to be known throughout the thesis. A solid understanding of camera functionality is essential for research in MR; therefore, the camera geometry is mathematically defined, and the digital image formation process is outlined. Furthermore, the basics of image synthesis are explained, focusing on creating virtual content, followed by an introduction to rendering and its mathematical framework in the rendering equation. Finally, a brief introduction to artificial neural networks in deep learning is given, as these networks are trained and applied in parts of the research.

Chapter 3 discusses sampling techniques as discrete approximations of continuous shapes. It addresses the importance of such techniques for 3D objects, with emphasis on applications where the surfaces or volumes of these objects are sampled with a discrete set of 3D points. The methods collected in [SS21] are presented in detail here.

Chapter 4 is dedicated to the real-time simulation of granular materials. It begins with a comprehensive review of current methods for simulating granular materials and highlights the challenges in achieving real-time performance. It describes the two-stage method proposed in [SSS22], comprising a low-resolution simulation with a higher computational cost and fewer elements and a high-resolution simulation with many elements and a lower computational cost, based on the results of the first simulation.

Chapter 5 focuses on one aspect of photometric registration: lighting estimation. It introduces the lighting representation proposed in [SSS23], including parameters such as light direction, light and ambient color, and a novel contrast level parameter. It then describes an artificial neural network trained to estimate these parameters from a single camera image. Additionally, the process of generating an appropriate training dataset for this network is explained. A comprehensive evaluation is presented to assess the quality of the trained network.

Chapter 6 addresses another aspect of photometric registration: the representation of shadows. It overviews existing methods for generating shadows—particularly cast shadows—and highlights their computational complexity. Then, a novel approach, as proposed in [SSS23], is introduced. This approach involves training tiny artificial neural networks to generate object-specific shadow textures based on the lighting parameters from Chapter 5. The result is a realistic cast shadow that can be efficiently generated on AR devices with minimal resource requirements. It further shows how to generate training data for the neural networks through rendering.

Chapter 7 summarizes the contributions made throughout this thesis and concludes with closing remarks.

2

Theoretical Background

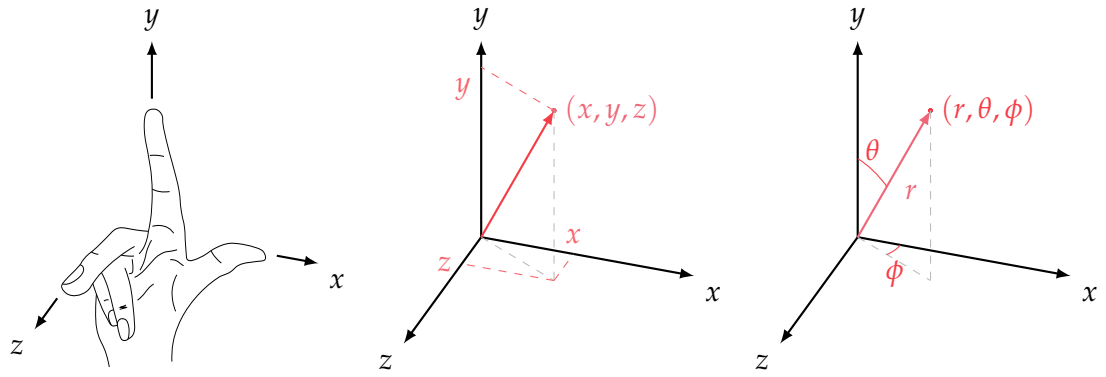
This chapter focuses on the theoretical foundation for the rest of the thesis. At this point, it should be mentioned that each subsequent chapter contains a separate, in-depth review of related work. In contrast, the background presented here briefly reviews basic concepts that may already be familiar to the experienced reader. However, at least the definition of a consistent notation in Section 2.1 should not be skipped. After that, Section 2.2 explains the functionality of a digital camera, its image formation process, and the model of its camera geometry. Section 2.3 shows how this image formation process is imitated when rendering virtual images. Finally, Section 2.4 outlines the structure and training of artificial neural networks and describes convolutional neural networks as an example for processing input images.

2.1 | Conventions and Definitions

It is common for different research disciplines to have their own historically evolved notations and conventions. Since this thesis integrates topics from many research areas, it is necessary to define a standardized notation at this point to ensure consistency throughout the thesis.

2.1.1 | Coordinate Systems

The definition of a coordinate system varies widely, especially in the orientation of the axes. While it is common in mathematics and physics to define the positive z-coordinate as the axis pointing up (i.e., opposite to gravity), it is common in computer graphics to define the positive y-coordinate as the *up vector*. In computer vision, on the other hand, it is typical for the x-y plane to be aligned with the image plane, resulting in the x-axis pointing to the right, the y-axis pointing down, and the z-axis aligned with the optical axis of the camera. Despite these different conventions, all of these definitions have in



(a) Right-hand rule (b) Cartesian (c) Spherical

Figure 2.1: (a): The relationship between the right-hand rule and the coordinate system chosen for this work. (b): A world point (in red) described by Cartesian coordinates in this coordinate system. (c): The same point (in red) as in (b) expressed in spherical coordinates.

common that they form a right-handed system, where the arrangement of the three axes corresponds to the same order as the thumb, index finger, and bent middle finger of the right hand (see Figure 2.1a). In this thesis, a right-handed coordinate system with the y-coordinate as the up vector is used, consistent with the dominant convention in computer graphics. A 3D Euclidean point in this coordinate system is referred to as a world coordinate and can be described by a Cartesian vector $\boldsymbol{x} = (x, y, z)^\intercal$ that points from the origin $\mathbf{0}$ of the coordinate system to the location of the point. Occasionally, for notational convenience, it is easier to enumerate individual axes rather than to refer to them as x, y, and z. The corresponding unit vectors describing these axes are then defined as:

$$\hat{\boldsymbol{e}}_1 = (1, 0, 0)^\intercal , \tag{2.1}$$

$$\hat{\boldsymbol{e}}_2 = (0, 1, 0)^\intercal , \tag{2.2}$$

$$\hat{\boldsymbol{e}}_3 = (0, 0, 1)^\intercal . \tag{2.3}$$

Some problems with rotational symmetry can be simplified by using spherical coordinates r, θ, ϕ instead of Cartesian coordinates. In this system, r represents the distance from the origin, θ is the polar angle, and ϕ is the azimuthal angle (see Figure 2.1c). Since the transformation from Cartesian coordinates to spherical coordinates

$$x = r \sin\theta \cos\phi , \tag{2.4}$$

$$y = r \cos\theta , \tag{2.5}$$

$$z = r \sin\theta \sin\phi , \tag{2.6}$$

and back

$$r = \sqrt{x^2 + y^2 + z^2}\,, \tag{2.7}$$

$$\theta = \operatorname{acos} y/r\,, \tag{2.8}$$

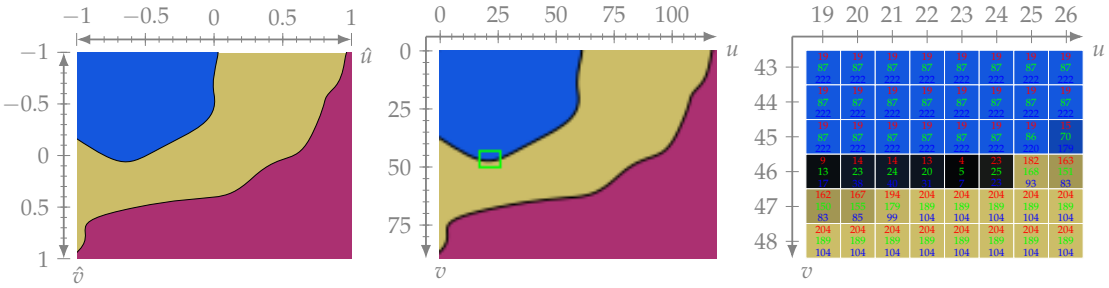
$$\phi = \operatorname{atan2}(z, x) \tag{2.9}$$

differs from the definition commonly used in mathematics for a z-up system, they are provided here for clarity. The relationship between the two coordinate systems is shown in Figure 2.1. The submanifold with $r = 1$, where all points lie on the unit sphere, is called *unit spherical coordinates*.

For many applications in computer graphics, affine transformations—including translations, rotations, scaling, and shearing—play a crucial role. In computer vision, projective geometry becomes important when describing projections of camera models, as will be demonstrated in Section 2.2.1. Both have in common that geometric operations can be expressed more elegantly using homogeneous coordinates. 3D homogeneous coordinates reside in the projective space $\mathbb{P}^3 = \mathbb{R}^4 - (0,0,0,0)^\mathsf{T}$. A homogeneous point can be described by a vector $\tilde{\boldsymbol{x}}_A \in \mathbb{P}^3$, from here on now denoted with the tilde operator. It is defined as:

$$\tilde{\boldsymbol{x}}_A = \begin{pmatrix} \boldsymbol{x}_A \\ 1 \end{pmatrix} = \begin{pmatrix} x_A \\ y_A \\ z_A \\ 1 \end{pmatrix}, \tag{2.10}$$

where $\boldsymbol{x}_A \in \mathbb{R}^3$ is the Cartesian vector of the corresponding Euclidean point in 3D space. All vectors $k \cdot (\boldsymbol{x}_A, 1)^\mathsf{T}$ with $k \neq 0$ form an equivalence class in $\mathbb{P}^3$ (i.e., representing the same point). Consequently, a homogeneous vector $\tilde{\boldsymbol{x}}_A$ has three degrees of freedoms (DoFs), namely the individual ratios of the four values. An advantage of homogeneous coordinates is their natural distinction between a point and a direction. Unlike points, directions are characterized by a zero in the final coordinate of the homogeneous vector. This distinction allows for consistent application of translations and scaling to both points and directions. Analogously, homogeneous coordinates exist for 2D points in $\mathbb{P}^2$. For this thesis, this brief introduction to homogeneous coordinates should suffice; for further details and their application in projective geometry, refer to [HZ04].



(a) (b) (c)

Figure 2.2: (a) shows a vector graphic with normalized image coordinates $\hat{u}$, $\hat{v}$. (b) shows a rasterized version of (a) with a resolution of 120×90 pixels with pixel coordinates u, v. (c) shows an enlarged section of (b), marked in green in (b), measuring 8×6 pixels, with the individual RGB values per pixel.

2.1.2 | Digital Images

We perceive the world around us as a three-dimensional space that can be described using a 3D coordinate system. An image displayed on a flat surface, such as a monitor, the display of an AR device, or in print, represents a projection of the 3D space onto a 2D surface. How this projection works is described in Section 2.2.1.

At this point, a definition of a (digital) image, also referred to as a texture in other contexts, shall be established first. In contrast to vector graphics (see Figure 2.2a) that describe images through mathematical paths and shapes, which can be resized arbitrarily without loss of quality, digital images (see Figure 2.2b), also referred to as rasterized images, are composed of a finite number of pixels. Each pixel stores brightness or so-called *intensity* information about different colors, respectively, channels. An image $\boldsymbol{I}$ has an integer width $w \in \mathbb{N}$ and height $h \in \mathbb{N}$, denoted as resolution $w \times h$. Each pixel in the image can be assigned a pixel coordinate

$$\boldsymbol{u} = \begin{pmatrix} u \\ v \end{pmatrix} \in \mathbb{N}_0^2, \tag{2.11}$$

where $u \in \{0, \dots, w-1\}$ corresponds to the pixel column, and $v \in \{0, \dots, h-1\}$ corresponds to the pixel row. In accordance with the prevalent definition in computer vision, the origin of the pixel coordinate system is located in the upper-left corner of the image. The u-axis points to the right, and the v-axis points down (see Figure 2.2).

In some cases, it is necessary to use continuous image coordinates instead of discrete pixel

coordinates. Again, there is a lack of consistent definitions for these coordinates. Within the scope of this thesis, these are denoted as normalized image coordinates $\hat{u}, \hat{v} \in [-1, 1]$ (see Figure 2.2a), with the origin in the center of the image.

Each pixel $\boldsymbol{I}(u, v)$ of the digital image $\boldsymbol{I}$ contains intensity information for a specified number c of channels. For a monochromatic image ($c = 1$), a single intensity channel is used, corresponding to luminance (i.e., the perceived brightness). Consequently, for an image in the RGB color space ($c = 3$), the intensities of the individual color channels (red R, green G, and blue B) result in a triplet for each pixel in the image matrix (see Figure 2.2c).

The intensities of individual channels are typically discretized. For conventional images, the intensity range is typically 8 bit, resulting in $2^8 = 256$ distinct values $\in \{0, \ldots, 255\}$. As discussed later in Section 5.2, different intensity ranges may be desired in some cases.

A special channel type is the alpha channel, used in image processing and virtual image synthesis. In this thesis, the alpha channel is encountered during the blending of virtual images with real images in AR applications. It provides information about the transparency of a pixel, with an alpha value of 0 indicating full transparency and a value of 255 in an 8 bit image indicating full opacity. However, alpha information cannot be captured in the image formation process of a real camera, which is described in the following section.

2.2 | The Digital Camera

The digital camera serves as the connecting component between the real environment and the virtual world in an AR application. To successfully develop such an application, it is essential to understand the functionality of a digital camera. The following section provides a fundamental description of the image formation process in digital cameras. This process involves several steps to create a digital 2D representation from the captured light of a 3D scene. Although the description refers to creating a single image, the principles apply equally to videos since they comprise a sequence of images.

The process begins when the camera's shutter opens for a preset amount of time—the exposure time—allowing light to enter the camera. The light is then directed towards the camera sensor through an arrangement of multiple lenses in the camera's *lens*. This arrangement varies widely depending on the lens design. The aperture is positioned within this lens arrangement and regulates the amount of light that passes through the

lens. The individual lens elements can be adjusted to bring the scene, or parts of the scene, in focus on the camera sensor. The size of the aperture is also indirectly related to the focus. A larger aperture allows more light to reach the sensor during the exposure time but reduces the depth of field (i.e., the distance range in which the scene is in focus). A smaller aperture, on the other hand, provides a greater depth of field. When light hits the camera sensor, it is captured at grid-arranged measurement points—the sensor pixels—and converted into electrical signals proportional to the light intensity. In a color image sensor, a color filter array—often a Bayer filter—is placed in front of the sensor to separate the incoming light into its red, blue, and green components. Due to the mosaic arrangement of the filter, each sensor pixel captures only one of these three color components. Each sensor pixel measures the intensity of the specific wavelength band corresponding to that color as an analog electrical current. The analog electrical signals are then converted to digital signals using an analog-to-digital converter, and the missing color information for each pixel is interpolated in a process known as demosaicing. For more details on the image formation process, refer to [GW18].

For many problems in computer vision, understanding the geometric properties of the image formation process is particularly important. Understanding camera geometry is the foundation for describing the relationships between 3D scenes and 2D images. It plays a crucial role in 3D reconstruction [HWH19], object tracking [TSS+15], pose estimation [TSS17], depth estimation [RSL+24], and image stitching [BL07]. In particular, it is essential for all AR applications since accurate camera geometry is a prerequisite for registering a virtual object precisely in a 3D scene. This allows camera movement without altering the virtual object's position in the scene.

2.2.1 | Camera Geometry

As mentioned earlier, the light rays pass through a varying arrangement of multiple lenses as they traverse the camera. Each lens refracts, reflects, and absorbs light in the process. Modeling all of these effects would result in a very complex system. However, for many purposes, it is sufficient to use a much simpler model known as the *pinhole model*. This model comprises a single theoretically infinitely small aperture—the pinhole—through which light is refracted. When using a pinhole camera, the entire scene appears sharp, analogous to a camera with a lens with a tiny aperture. Therefore, the pinhole model is suitable for adequately modeling a well-focused camera with reasonable accuracy. This approximation is sufficient for many applications in computer vision. It comprehensively describes how points in 3D space are mapped to pixels in a 2D image

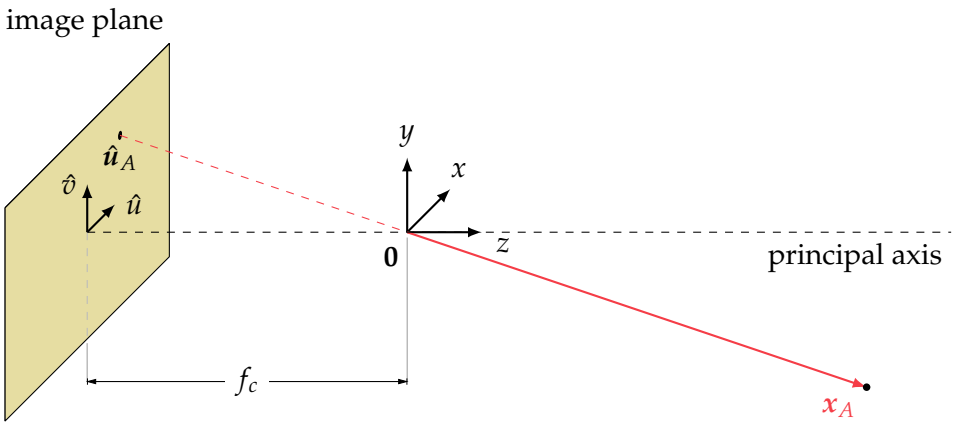


Figure 2.3: A schematic representation of the pinhole camera geometry. The camera has its center at the origin $\mathbf{0}$ of the 3D coordinate system. A 3D point $\boldsymbol{x}_A$ is projected onto the image plane at a distance f_c behind the camera center.

by perspective projection.

Here, the camera center, also known as the optical center of the pinhole camera—in the following just the *camera*—is positioned at the origin $\mathbf{0}$ of the 3D coordinate system. The camera is oriented so that its line of sight, denoted as the *principal axis*, points in the direction of the z-axis. The plane on which the image is projected—the *image plane*—is located at a distance $z = -f_c$. The distance f_c is called the *focal length*. The point where the principal axis intersects the image plane is called the *principal point* and is the origin of the normalized image coordinate system $(\hat{u}, \hat{v})$. The image plane is aligned with the position and orientation of the sensor in the digital camera. The complete configuration is shown in Figure 2.3.

For a point $\boldsymbol{x}_A$ to appear in the image, a ray of light must pass through the infinitesimal aperture in the camera center at $\mathbf{0}$ and intersect the image plane. By examining the ratios of the resulting triangles in Figure 2.3, the coordinates $(-f_c x_A/z_A, -f_c y_A/z_A, f_c)^\mathsf{T}$ for the point on the image plane can be determined. This corresponds to a point

$$\hat{\boldsymbol{u}}_A = \begin{pmatrix} -f_c x_A/z_A \\ -f_c y_A/z_A \end{pmatrix} \tag{2.12}$$

in the normalized image coordinates of the image plane. The resulting image is mirrored in both axes. Flipping both axes $(\hat{u}_A, \hat{v}_A) \to (-\hat{u}_A, -\hat{v}_A)$ results in the orientation of the image axes introduced in Section 2.1.2.

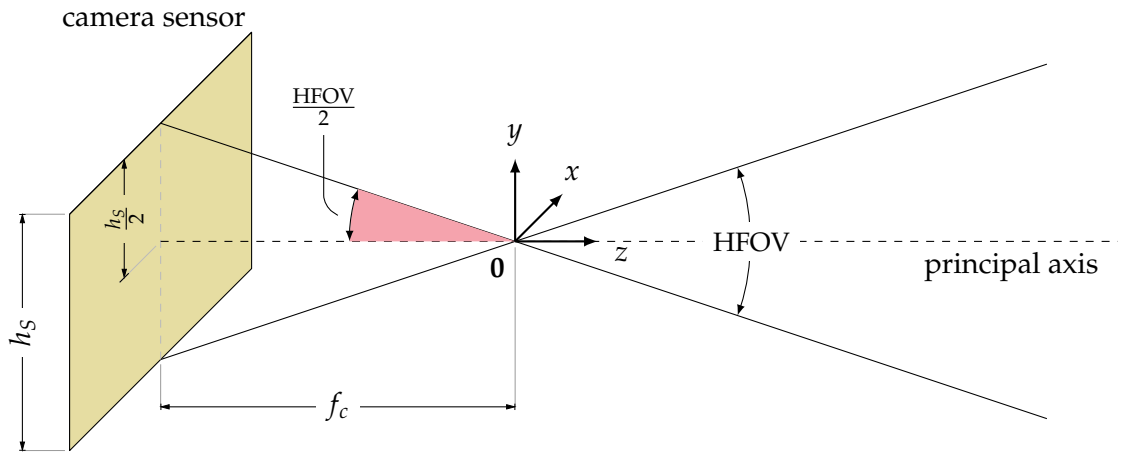


Figure 2.4: A visualization of the geometric relationships between horizontal field of view (HFOV), camera sensor height h_S, and focal length f_c.

For practical reasons, when working with digital images, it is often beneficial to know the location of a point in the image in pixel coordinates (u, v). This can be achieved by expressing the metric quantity of the focal length f_c in terms of pixels. This requires knowledge of the pixel density of the sensor. Sensor pixels may have a non-square size. Consequently, a distinction is made between pixels per unit length in the u-direction ρ_u and the v-direction ρ_v. This results in two different focal lengths expressed in terms of pixels $f_u = \rho_u f_c$ and $f_v = \rho_v f_c$. As described in Section 2.1.2, the origin of the pixel coordinate system is usually located in the upper left corner of the image and not at the principal point. Therefore, the coordinates must be adjusted accordingly. For a point $\hat{\boldsymbol{u}}_A$ in normalized image coordinates, a point $\boldsymbol{u}_A$ in pixel coordinates is given by:

$$\boldsymbol{u}_A = \begin{pmatrix} f_u x_A / z_A \\ f_v y_A / z_A \end{pmatrix} + \begin{pmatrix} c_u \\ c_v \end{pmatrix}, \tag{2.13}$$

where $(c_u, c_v)^\mathsf{T}$ is the principal point in pixel coordinates.

The focal length is not a directly interpretable measure since camera systems with different sensor sizes capture different portions of the same scene with the same focal length. It is more natural to express an angular field of view (FOV). The FOV is a technical analog to the viewing angle in human perception for optical instruments (e.g., cameras). Figure 2.4 shows the relationship between the focal length f_c, the camera sensor height h_S, and the horizontal FOV. Using trigonometric principles yields:

$$\tan\left(\frac{\text{HFOV}}{2}\right) = \frac{h_S/2}{f_c}. \tag{2.14}$$

The ratio between the two quantities holds for their analogous pixel quantities image height h and focal length f_v. For square pixels (where $f_u = f_v$) this relationship also holds in the diagonal, resulting in focal lengths

$$f_u = f_v = \frac{\sqrt{h^2 + w^2}}{2} \frac{1}{\tan\left(\frac{\text{FOV}}{2}\right)} \tag{2.15}$$

in terms of pixels, where FOV is the diagonal angular field of view. Therefore, a relationship between FOV, image width w, image height h, and focal length in pixels f_u and f_v can be established.

Expressing the image point $\boldsymbol{u}_A$ in homogeneous coordinates $\tilde{\boldsymbol{u}}_A = (\boldsymbol{u}_A, 1)^\intercal \in \mathbb{P}^2$ allows for a concise representation of the entire projection in matrix form. As described in Section 2.1.1, a point $z_A(\boldsymbol{u}_A, 1)^\intercal$ with $z_A \neq 0$ is part of the equivalence class of $(\boldsymbol{u}_A, 1)^\intercal$, so it represents the same point in space. This yields:

$$z_A \begin{pmatrix} u_A \\ v_A \\ 1 \end{pmatrix} = z_A \begin{pmatrix} f_u x_A / z_A + c_u \\ f_v y_A / z_A + c_v \\ 1 \end{pmatrix} = \begin{pmatrix} f_u x_A + c_u z_A \\ f_v y_A + c_v z_A \\ z_A \end{pmatrix} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}, \tag{2.16}$$

where the 3x3 matrix

$$\boldsymbol{K} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

is called the *camera calibration matrix* or intrinsics matrix. The intrinsic parameters of the camera f_u, f_v, c_u, c_v, along with additional parameters accounting for lens distortion effects can be determined by a process called *camera calibration*. Camera calibration algorithms are included in standard computer vision libraries such as OpenCV[1]. For more details on camera calibration algorithms, refer to [Z00; HZ04].

In applications where the camera is not static (e.g., AR), it is still desirable to have a static coordinate system in which the spatial position of points can be described independently of the current camera pose. For this purpose, it is beneficial to distinguish between two 3D coordinate systems: a camera coordinate frame and a world coordinate frame. The camera coordinate system is, as described earlier, positioned with the camera center

[1] OpenCV is available at: `https://opencv.org/`

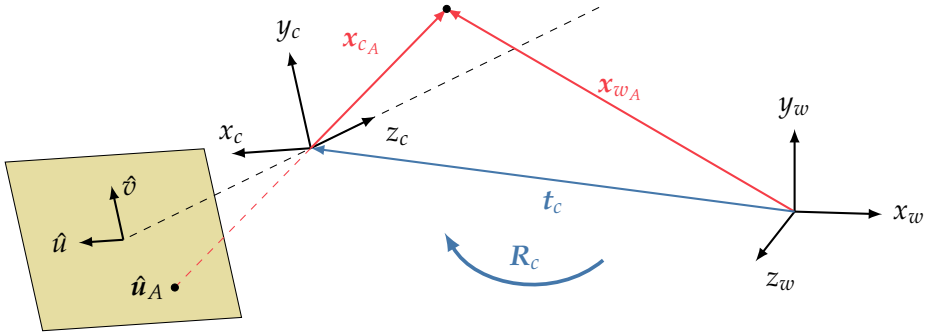


Figure 2.5: A point represented in an absolute world coordinate system and a relative camera coordinate system. The relationship between the two coordinate systems is described by a translation vector $\boldsymbol{t}_c$ and a rotation matrix $\boldsymbol{R}_c$. The point is projected from the camera coordinate system onto the image plane, analogous to Figure 2.3.

at the origin and the camera's line of sight directed along the z-axis. The absolute world coordinate system is connected to the camera coordinate system by an Euclidean transformation, specifically a rotation $\boldsymbol{R}_c$ and a translation $\boldsymbol{t}_c$ (see Figure 2.5). When the camera pose changes, this transformation will also change. Consequently, an image point $\boldsymbol{u}_A$ in the camera image describes the same point $\boldsymbol{x}_{c_A}$ in the camera coordinate system for both poses but different points $\boldsymbol{x}_{w_A}$ and $\boldsymbol{x}'_{w_A}$ in the world coordinate system. When expressing a world coordinate point $\boldsymbol{x}_{w_A}$ in homogeneous coordinates $\tilde{\boldsymbol{x}}_{w_A} = (\boldsymbol{x}_{w_A}, 1)^\intercal \in \mathbb{P}^3$, the transformation into the camera coordinate system

$$\boldsymbol{x}_{c_A} = \begin{bmatrix} \boldsymbol{R}_c & \boldsymbol{t}_c \end{bmatrix} \tilde{\boldsymbol{x}}_{w_A} \tag{2.18}$$

can be described by a 3x4 matrix called the *camera extrinsics*. The camera extrinsics comprise a 3x3 rotation matrix $\boldsymbol{R}_c$ and a 3x1 translation vector $\boldsymbol{t}_c$. The parameters of $\boldsymbol{R}_c$ and $\boldsymbol{t}_c$ are referred to as external parameters or exterior orientation and describe the orientation and position of the camera. Using the previously described camera intrinsics matrix $\boldsymbol{K}$, a point can be directly transformed from homogeneous world coordinates $\tilde{\boldsymbol{x}}_{w_A}$ to homogeneous pixel coordinates $\tilde{\boldsymbol{u}}_A$ with:

$$\tilde{\boldsymbol{u}}_A = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R}_c & \boldsymbol{t}_c \end{bmatrix} \tilde{\boldsymbol{x}}_{w_A} . \tag{2.19}$$

The matrix $\boldsymbol{P} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R}_c & \boldsymbol{t}_c \end{bmatrix}$ is called the *camera matrix* or projection matrix.

2.3 | Virtual Image Synthesis

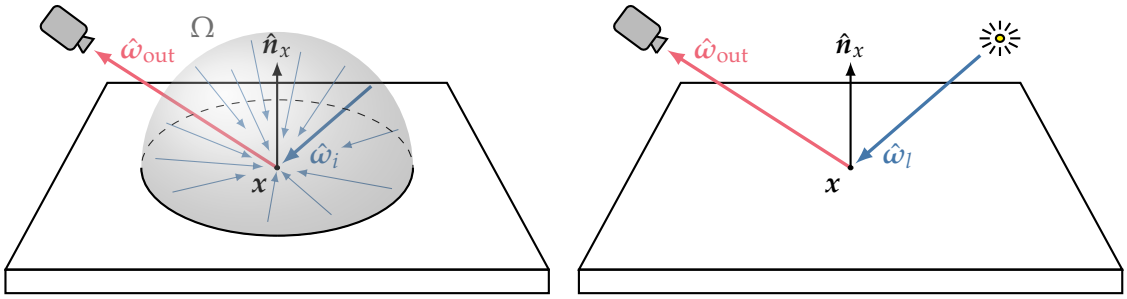
In addition to the digital image formation process described earlier, which creates a digital representation of the real world, another image formation process called virtual image synthesis plays an important role in AR applications. This process—also known as *rendering*—describes how a virtual scene or parts of it are transformed into an image. In non-see-through AR applications, the camera and the virtual images are merged into a single image. Since rendering is a vast research area, only the terms and concepts most relevant to this thesis will be briefly outlined here, with references for more in-depth information.

2.3.1 | Rendering Equation

To create a realistic image of a virtual scene, it is necessary to model the light transport in a physically correct way. Several models exist to conceptualize light transport, each with a different level of detail. Quantum electrodynamics [F85] is the most detailed model, allowing the simulation of effects such as fluorescence and phosphorescence. Classical electromagnetic theory [G17], based on Maxwell's equations, follows as a less detailed but still comprehensive model, allowing the simulation of polarization and dispersion. A further simplification is wave optics [A20], which captures phenomena such as diffraction and interference. For most effects, however, it is sufficient to use a simpler and more computationally efficient model: ray optics [H17]. This model describes the light propagation along rays and its interaction through emission, reflection, and transmission. To produce color renderings using this model, the continuous wavelength distribution of the simulated light is typically divided into three discrete wavelength bands corresponding to the RGB colors.

Radiometric quantities are used to characterize light transport in physical units. In physically-based rendering, the quantity of interest is radiance $L^{(e)}$. Radiance measures the light energy that strikes or emits from a given area in a given direction. Its SI unit $[\mathrm{W}/(\mathrm{m}^2 \cdot \mathrm{sr})]$ describes the power of electromagnetic radiation per solid angle per unit area. An important principle in ray optics is the radiance invariance law, which states that the radiance along a ray in an ideal optical system remains invariant. In rendering, air is commonly approximated as such an ideal optical system. This assumption allows for formulating the rendering equation [K86; ICG86]

$$L_{\mathrm{out}}^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\mathrm{out}}) = L_e^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\mathrm{out}}) + \int_{\Omega} f_{\mathrm{BRDF}}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\mathrm{out}}, \hat{\boldsymbol{\omega}}_i) L_{\mathrm{in}}^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_i)(-\hat{\boldsymbol{\omega}}_i^{\mathsf{T}} \hat{\boldsymbol{n}}_x) \mathrm{d}\hat{\boldsymbol{\omega}}_i \,, \tag{2.20}$$



(a) Hemispherical evaluation (b) Direct lighting approximation

Figure 2.6: For a point $\boldsymbol{x}$, the exitant radiance $L_{\text{out}}^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\text{out}})$ in direction $\hat{\boldsymbol{\omega}}_{\text{out}}$ is determined by solving the hemispherical integral of the rendering equation in (a) and by considering only of the direct lighting contribution of a point light source in (b).

which is the foundation of physically accurate rendering. It describes the outgoing or exitant radiance $L_{\text{out}}^{(e)}$ in the direction $\hat{\boldsymbol{\omega}}_{\text{out}}$ at a point $\boldsymbol{x}$. The term $L_e^{(e)}$ represents the self-emitted radiance of that point in the direction $\hat{\boldsymbol{\omega}}_{\text{out}}$. The integral over the hemispherical solid angle Ω accumulates all incoming or incident radiance values $L_{\text{in}}^{(e)}$ from all directions $\hat{\boldsymbol{\omega}}_i$ within Ω (see Figure 2.6a). The function f_{BRDF} represents the bidirectional reflectance distribution function (BRDF) [N65; NRH+77], which describes how the surface at point $\boldsymbol{x}$ with surface normal $\hat{\boldsymbol{n}}_x$ reflects light. It depends on several factors, such as surface geometry and material properties. For more detailed information about the BRDF, refer to [GGG+16]. To generate a virtual image, the light transport is simulated using the rendering equation to calculate the irradiance that a physical camera sensor at the location of the virtual camera would measure.

The incident radiance term $L_{\text{in}}^{(e)}$ in the hemispherical integral of the rendering equation corresponds to the exitant radiance $L_{\text{out}}^{(e)}$ of other points hitting point $\boldsymbol{x}$ from the direction $\hat{\boldsymbol{\omega}}_i$. Since all these surface points influence each other with their exitant radiance, the solution of the rendering equation is infinitely recursive and therefore usually cannot be evaluated exactly analytically. Different rendering methods, therefore, aim to approximate this equation more or less exactly. There is often a trade-off between speed, noise, and accuracy. It should be mentioned that some rendering methods predate the formulation of the rendering equation itself but can still be derived from it.

2.3.2 | Direct Lighting

A very simple approximation is to consider only the incident radiance contributions of light sources according to their self-emitting radiance $L_e^{(e)}$:

$$L_{\text{out}}^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\text{out}}) = \sum_{\text{Lights } i} \int_{\Omega} f_{\text{BRDF}}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\text{out}}, \hat{\boldsymbol{\omega}}_j) L_{e_i}^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_j)(-\hat{\boldsymbol{\omega}}_j^{\mathsf{T}} \hat{\boldsymbol{n}}_x) \mathrm{d}\hat{\boldsymbol{\omega}}_j \,. \tag{2.21}$$

This is called *direct lighting*. Neglecting the spatial extent of the light sources eliminates the need for the hemispherical integrals in Equation (2.21). It leaves only the summation over all light sources in the scene:

$$L_{\text{out}}^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\text{out}}) = \sum_{\text{Lights } i} f_{\text{BRDF}}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{\text{out}}, \hat{\boldsymbol{\omega}}_{l_i}) L_{e_i}^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{l_i})(-\hat{\boldsymbol{\omega}}_{l_i}^{\mathsf{T}} \hat{\boldsymbol{n}}_x) \Theta_v(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{l_i}) \,, \tag{2.22}$$

where $\hat{\boldsymbol{\omega}}_{l_i}$ is the normalized direction from the light source to point $\boldsymbol{x}$ and $\Theta_v(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_{l_i})$ is the visibility term, which returns 1 if the light source is directly visible from point $\boldsymbol{x}$ and 0 if an object obscures the direct path. Figure 2.6 illustrates the entire rendering equation in 2.6a and the simplification to direct lighting in 2.6b.

In computer graphics, there are several types of light sources. Point lights, directional lights, and area lights are the most important and relevant for this thesis. The point light with a total radiant power of Φ_p emits light uniformly in all directions from its location $\boldsymbol{x}_p$, similar to a light bulb. It has an incident radiance contribution

$$L_{e_p}^{(e)}(\boldsymbol{x}) = \frac{\Phi_p}{4\pi |\boldsymbol{x} - \boldsymbol{x}_p|^2} \tag{2.23}$$

at a surface point $\boldsymbol{x}$.

The directional light, on the other hand, emits light uniformly in only one direction. It is treated as a distant light source, similar to the sun. Therefore, all points in the scene can be considered to be approximately equidistant from the light source, resulting in a uniform radiance contribution $L_{e_d}^{(e)}$ from the directional light throughout the scene.

Unlike the previous two types of lights, area lights are not infinitesimally small objects but physical shapes that emit light. As a result, they produce softer shadows and more realistic lighting than the previous two. However, calculating their radiance contribution is more complex because there is no single direction $\hat{\boldsymbol{\omega}}_{l_i}$ pointing from point $\boldsymbol{x}$ to the light source. To determine the contribution of an area light, the hemispherical integral in Equation (2.21) must be evaluated. This can be simplified by reformulating the integral over the hemisphere into an integral over the area of the light source itself. Therefore,

it is necessary to describe the relationship between the differential area element of the surface of the area light and the differential area element of the hemisphere:

$$\mathrm{d}\hat{\boldsymbol{\omega}}_j = \frac{(\hat{\boldsymbol{\omega}}_y^\mathsf{T}\hat{\boldsymbol{n}}_y)}{|\boldsymbol{x}-\boldsymbol{y}|^2}\mathrm{d}A\,. \tag{2.24}$$

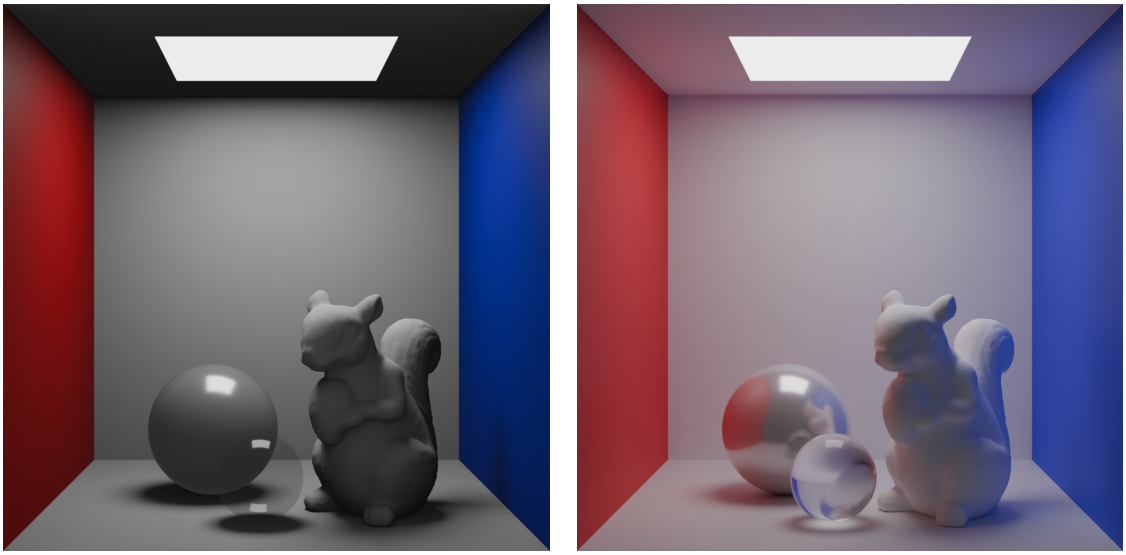
The variable $\boldsymbol{y}$ corresponds to the points on the surface A with surface normals $\hat{\boldsymbol{n}}_y$, and $\hat{\boldsymbol{\omega}}_y$ is the normalized direction from point $\boldsymbol{y}$ to point $\boldsymbol{x}$. For more information on the relationships between different differential area elements—also known as *view factors*—refer to [IDB+17]. Therefore, it is not necessary to calculate the entire hemispherical integral, but only the integral

$$\frac{\Phi_a}{A}\int_A f_{\mathrm{BRDF}}(\boldsymbol{x},\hat{\boldsymbol{\omega}}_{\mathrm{out}},\hat{\boldsymbol{\omega}}_y)\frac{(-\hat{\boldsymbol{\omega}}_y^\mathsf{T}\hat{\boldsymbol{n}}_x)(\hat{\boldsymbol{\omega}}_y^\mathsf{T}\hat{\boldsymbol{n}}_y)}{|\boldsymbol{x}-\boldsymbol{y}|^2}\Theta_v(\boldsymbol{x},\hat{\boldsymbol{\omega}}_y)\mathrm{d}A \tag{2.25}$$

over the surface A of the area light, where Φ_a is the total radiant power of the area light. Evaluating this integral can still be computationally expensive, especially for more complex light source geometries. In interactive applications, several further approximations are often used to efficiently calculate the direct lighting contribution of area light sources while maintaining reasonable accuracy. For further information on real-time rendering strategies, refer to [AHH+18].

Direct lighting is often used with simple lighting models such as Phong [P75] or Blinn-Phong [B77]. These models approximate the BRDF using simple, empirically derived functions. The Phong model expresses the BRDF as a combination of three terms: an ambient term, which provides uniform illumination regardless of the position of the light source due to the overall ambient color of the scene; a diffuse term, which describes the reflection of light from rough surfaces that scatter radiance uniformly in all directions, depending only on the surface normal and the direction to the light source $\boldsymbol{\omega}_{l_i}$; and a specular term, which accounts for specular highlights on shiny surfaces and depends not only on the direction to the light source and the surface normal but also on the viewing direction. The Blinn-Phong model improves on the Phong model by more accurately approximating of the specular highlight using the halfway vector between the view and light directions.

A widely used method for calculating direct lighting contributions is ray tracing [A68]. This involves casting rays from each pixel in the image plane through the camera center into the scene (recall Section 2.2.1). At each point where the ray intersects a surface, the direct lighting contribution (Equation (2.22)) is evaluated (see Figure 2.7a). The recursive ray tracing algorithm [W79]—also known as Whitted ray tracing—further casts



(a) Ray traced direct lighting (b) Path traced global illumination

Figure 2.7: A scene with an opaque diffuse squirrel, a glossy metallic sphere, and a transparent glass sphere; rendered with direct lighting using ray tracing in (a); rendered with full global illumination using path tracing in (b).

one or more secondary rays from the intersection point, if the surface is reflective or transparent, to simulate reflections and refractions. These secondary rays are recursively traced through the scene, and their direct lighting contributions are evaluated at the next intersection point.

2.3.3 | Global Illumination

Direct lighting, as previously described, can effectively capture effects such as highlights and shadows, providing important visual cues about the relative positions and shapes of objects in a scene. However, it cannot describe complex light transport phenomena that occur when light bounces and scatters multiple times within a scene. These effects—known as *indirect lighting*—result from including the exitant radiance of all surfaces in the hemispherical integral of the rendering equation (Equation (2.20)). Indirect lighting is crucial for achieving realistic image synthesis. It can be used to calculate more complex phenomena such as ambient occlusion, which simulates uniform shadowing in corners; caustics, a pattern of highlights on surfaces in the scene created when light is reflected or refracted off curved surfaces; or color bleeding, where colors from one surface are cast onto a second surface (see the squirrel in Figure 2.7b). For realistic rendering,

it is therefore necessary to describe both direct and indirect lighting, known as *global illumination*.

Just as pure direct lighting can be understood as a very simple approximation of the rendering equation, global illumination as a whole can also be approximated by various techniques. One of these techniques is image-based lighting (IBL) [D98], where the scene's radiance information is stored in an environment map. The key point of IBL is the equidistant scene assumption, which implies that every point in the environment map is infinitely far away from the viewpoint from which it was captured. This means that the incident radiance $L_{\text{in}}^{(e)}$ at this point can be looked up directly from the environment map based on the direction $\hat{\boldsymbol{\omega}}_i$. However, this assumption can lead to significant inaccuracies, especially in scenes with many nearby objects, because the environment map can vary significantly between two viewpoints. One way to improve these inaccuracies is to use multiple environment maps from different viewpoints, called light probes. During rendering, the incident radiance $L_{\text{in}}^{(e)}$ for a point $\boldsymbol{x}$ can be determined by interpolating the radiance values of the environment maps of the nearest light probes. Light probes can be captured in the real world, as described in more detail in Section 5.4.1. However, they can also be computed synthetically by tracing rays from the probe location into the virtual scene and accumulating the incident radiance. The number and placement of such light probes within a scene always involve a trade-off between computational complexity and accuracy.

IBL can be accelerated by approximating environment maps with a series expansion on spherical basis functions. The two most commonly used basis functions for this purpose are spherical harmonics [SKS02] and spherical Gaussians [TS06]. Typically, a low-order series expansion is used to describe the environment map with only a few coefficients. For example, a second-order spherical harmonics approximation requires only four coefficients per color channel, and a third-order approximation requires nine coefficients. The representation resulting from a low-order series expansion of the environment map lacks detail. It is therefore unsuitable for realistically describing reflections on specular surfaces, but it is suitable for approximating spatially varying lighting.

IBL efficiently approximates ambient lighting and environment reflections, but it faces challenges with shadows and occlusions. The equidistant scene assumption fails to account for occlusions caused by nearby geometry, making it impossible to cast a shadow from an object onto a surface with pure IBL. Additionally, static and pre-computed environment maps lack the ability to simulate dynamic shadows. There are other methods that overcome these limitations by approximating the rendering equation more

accurately, but they are also computationally more expensive.

Path tracing [K86] is the method that can most accurately approximate the rendering equation, allowing for the most realistic image synthesis. It can be thought of as an extended form of ray tracing. In backward path tracing, rays are shot from the camera into the scene (similar to ray tracing); in forward path tracing, rays are shot from light sources; and in bidirectional path tracing [LW93], rays are shot from both the camera and the light sources. Unlike ray tracing, path tracing uses stochastic sampling techniques—known as Monte Carlo methods—to create random paths at the starting and intersection points. Consequently, path tracing produces different results each time, which converge to the correct solution as the number of evaluated paths approaches infinity. To simulate global illumination, the entire rendering equation is evaluated at each intersection point, recursively accumulating the radiance contributions through all randomly generated paths originating at that point. In practice, the tracing of a path is stopped after a certain number of maximum bounces (e.g., to avoid an infinite recursive loop). Due to stochastic sampling, path tracing initially produces a very noisy image, with the noise decreasing as the number of evaluated paths increases. Since path tracing requires the evaluation of many paths, it is not only the most accurate of the presented approximations of the rendering equation but also the most computationally expensive. Figure 2.7 shows a comparison of a scene with three objects rendered with direct lighting using ray tracing on the one hand and with global illumination using path tracing on the other hand.

It is worth mentioning that path tracing can also be used in combination with environment maps. This approach—known as *environment sampling*—has the advantage that it is not necessary to build the entire virtual scene with geometry but rather describe the far field of the scene by an environment map. If a ray misses all the geometry in the near field of the scene and intersects the environment map, this path does not need to be terminated. Instead, the radiance can be sampled from the environment map [RTJ95; ARB+03]. For additional information on physically correct rendering, refer to [PJH23].

2.4 | Machine Learning

As mentioned earlier, understanding the real environment using the camera image plays an important role in AR. In the early days of AR, classical algorithmic computer vision methods were used for this task. Nowadays, methods based on artificial intelligence (AI) have replaced these classical techniques and excel in critical tasks for AR applications, such as feature detection [XCX+24], plane detection [XGY+22], photometric registration [AT20], and image segmentation [MBP+22]. Therefore, a basic understanding of AI—more specifically machine learning (ML)—is essential for AR applications. This section provides a brief overview of the most relevant concepts for understanding the AI-based methods presented later.

ML is a superordinate term for a subfield of AI. It includes all methods that analyze and learn from data to make better decisions. Early ML methods were designed to detect hand-crafted features or patterns in data. Alongside other early AI techniques, these knowledge-crafted methods peaked during the first wave of AI that ended in the late 1980s. The field of ML gained popularity during the ongoing second wave of AI with the advent of *deep learning*.

Deep learning relies on deep artificial neural networks, which are described in more detail below. These networks automatically *learn* the features or patterns in data necessary to solve a problem, eliminating the need for manual feature engineering used in earlier ML algorithms. The network parameters are optimized using statistical models, which is why the second wave of AI is driven by statistical learning. However, learning features automatically require a significant amount of data and computational resources. The availability of increasingly larger datasets and more computing power is a second driver for the renewed hype around AI.

In the context of this thesis, the term ML is used synonymously with the term deep learning. A detailed examination of the machine learning methods used before deep learning can be found in [B06]. Further information on deep learning can be found in [GBC16]. For a comprehensive overview of the history of AI, refer to [W21].

2.4.1 | Artificial Neural Networks

An artificial neural network is based on a simplified model of the human brain, with its basic components being artificial neurons. In the human brain, a neuron transmits its action potential to connected neurons when it is sufficiently stimulated to become activated. In current models of artificial neurons, activation is characterized by an output signal represented by a floating-point number. Activation is determined by a differentiable, nonlinear, non-constant function associated with the neuron, known as the activation function f_α. This nonlinearity is crucial for modeling the complex nonlinear relationships in real-world data. Without the nonlinearity, the network would be limited to producing a linear transformation of the input variables. The activation functions f_α for an input $\chi \in \mathbb{R}$ that are important for this work are the rectified linear unit (ReLU)

$$\mathrm{ReLU}(\chi) := \begin{cases} \chi & \text{if} \quad \chi \geq 0 \\ 0 & \text{else}\,, \end{cases} \tag{2.26}$$

the exponential linear unit (ELU) [CUH16]

$$\mathrm{ELU}(\chi) := \begin{cases} \chi & \text{if} \quad \chi > 0 \\ e^\chi - 1 & \text{else}\,, \end{cases} \tag{2.27}$$

the sigmoid function

$$\sigma_\alpha(\chi) := \frac{1}{1+e^{-\chi}}\,, \tag{2.28}$$

and the hyperbolic tangent (tanh)

$$\tanh(\chi) = \frac{e^\chi - e^{-\chi}}{e^\chi + e^{-\chi}}\,. \tag{2.29}$$

Neurons are organized in parallel sets called *layers*, and neurons in different layers can have connections. Each of these connections has a multiplicative factor called weight w_{nn}. The activation

$$\alpha_{\mathrm{nn}_k} = f_\alpha\left(\chi = \beta_{\mathrm{nn}_k} + \sum_i w_{\mathrm{nn}_i}\alpha_{\mathrm{nn}_i}\right) \tag{2.30}$$

of a k-th neuron is the output of the activation function f_α for the linear combination of all weighted neuron connections i connected to the k-th neuron, plus optionally an additive bias β_{nn_k}.

One of the simplest forms of artificial neural networks is a fully connected multi-layer perceptron (MLP). It comprises several layers in which all neurons are connected to all neurons in the next layer. Layers with this configuration are called fully connected layers

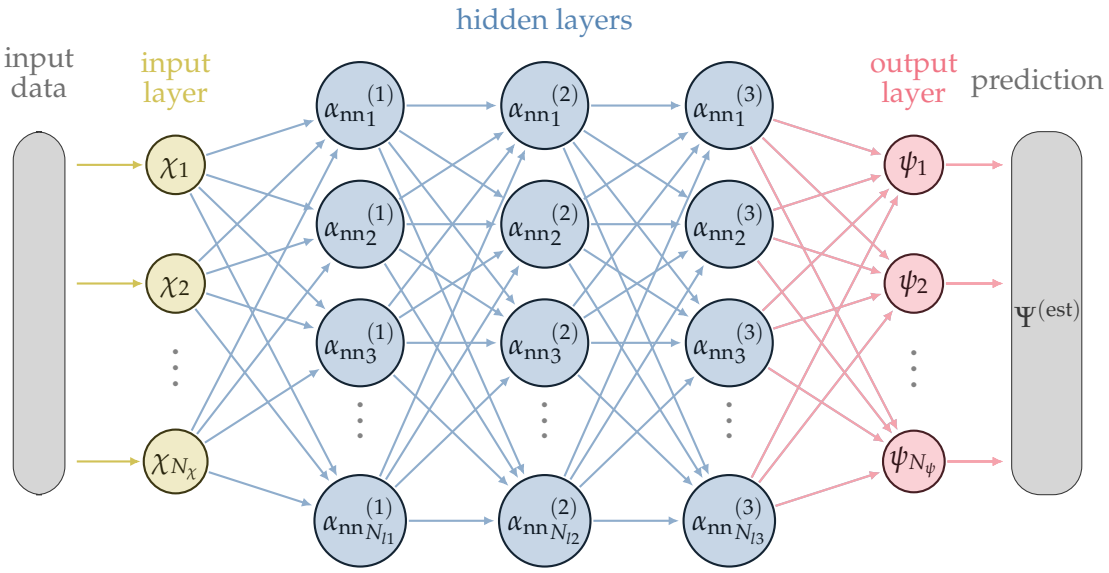


Figure 2.8: An example of a multi-layer perceptron with N_χ neurons as the input layer, three hidden layers with respectively N_{l1}, N_{l2}, and N_{l3} neurons and N_ψ neurons as the output layer.

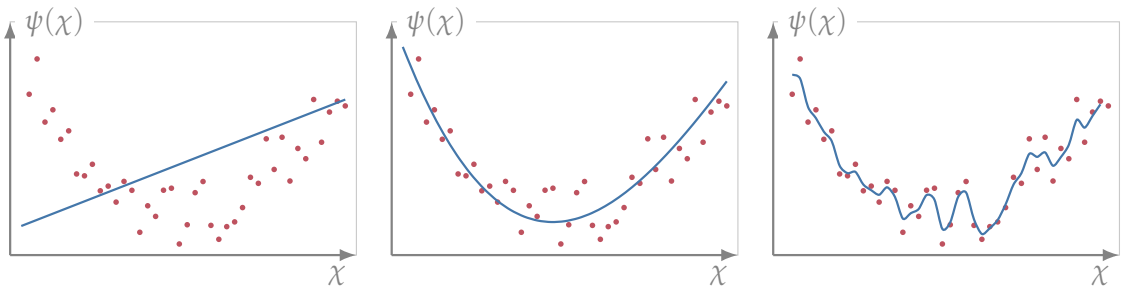
or dense layers. The neurons in the first layer are called input neurons $\boldsymbol{\chi}$, and those in the last layer are called output neurons $\boldsymbol{\psi}$. The layers in between are the hidden layers. Figure 2.8 shows a simple MLP with three hidden layers.

The output $\Psi^{(\mathrm{est})} = \boldsymbol{\psi}$ of a neural network is called the *prediction* or estimate. Optimizing such a network requires the ability to evaluate its performance. If the actual result $\Psi^{(\mathrm{gt})}$—also called the *ground truth*—for a given input is known, it can be compared with the prediction. For this purpose, a penalty is applied, referred to as cost or loss $\mathcal{L}$ in ML. A high loss corresponds to a significant deviation from the ground truth, while a loss of 0 corresponds to a match between ground truth and prediction. One of the most commonly used loss functions is the mean squared error (MSE) loss

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{N_{\mathrm{td}}} \sum_{i=1}^{N_{\mathrm{td}}} \left(\Psi_i^{(\mathrm{gt})} - \Psi_i^{(\mathrm{est})} \right)^2 , \tag{2.31}$$

where N_{td} is the total number of prediction and ground truth data pairs.

Training a neural network involves determining the optimal parameters of the network—the weights and biases—for a given problem. This optimization process can be automated but requires a substantial amount of data, referred to as training data. Training data typically comprises input samples (e.g., images) and corresponding labels that represent the expected output values of the network (e.g., classes in classification



(a) Underfitting (b) Appropriate fitting (c) Overfitting

Figure 2.9: A visualization of underfitting (a), appropriate fitting (b), and overfitting (c) on the same data distribution due to different model complexities.

problems). It should be noted that there are cases where neural networks are trained on unlabeled data, a technique known as unsupervised learning. Unsupervised learning can help identify unknown patterns or correlations in unlabeled data. For a more comprehensive overview of unsupervised learning, refer to [NAA+23].

The optimal size N_{td} of the training dataset depends on the complexity of the model (i.e., the number of parameters in the network). If the dataset is too large or the model is too simple, the network cannot accurately capture the variances in the data, leading to *underfitting* (see Figure 2.9a). Conversely, suppose the dataset is too small or the model is too complex. In that case, the network may learn to reproduce the training data but fail on new, unseen data—a phenomenon known as *overfitting* (see Figure 2.9c). Figure 2.9 illustrates this concept by fitting different model complexities to the same data.

The dataset is divided into three segments: training data, validation data, and test data to evaluate the performance of a model on unseen data. The training data is used only to optimize the model parameters. During training, the validation data assesses how well the current model generalizes, indicating its performance on unseen data. Unlike the training loss, the validation loss is not used to optimize the model parameters. After all optimizations, the test data is used only to evaluate the final model. Test scores are used to avoid optimizing the model toward the best validation loss. They are also crucial for comparing the model's performance against different architectures.

The process of finding optimal parameters for the network using the training data is called *backpropagation*. It is an automated optimization process aiming to minimize the loss. The negative gradient of the loss function defines how the parameters must be changed. Therefore, it is essential that the activation functions are differentiable;

otherwise, the gradient of the prediction $\Psi^{(\mathrm{est})}$ cannot be calculated. Gradients with respect to parameters in one layer depend on those in subsequent layers. Therefore, the gradient flow proceeds sequentially from layer to layer using the chain rule. This process is called backpropagation because the error propagates backward through the network.

When dealing with large amounts of training data, as typical in deep learning, it is impractical or even impossible to form the loss gradient for all training samples. In practice, a randomly selected small subset of the training data called a *mini-batch*, is used. The size of this subset is the mini-batch size or batch size. An *epoch* concludes when the network has been trained on enough mini-batches to encounter every element of the original training data. The gradients of the mini-batch loss concerning each individual network parameter ρ_{nn}—the weights w_{nn} and biases β_{nn}—are used to minimize the total loss using stochastic gradient descent. The gradient defines the direction in which a parameter ρ_{nn} is optimized toward a minimum. The step size by which the parameter ρ_{nn} approaches the minimum is defined by a scalar parameter λ_{lr}, called *learning rate*. This results in the following update rule for the parameter:

$$\rho_{\mathrm{nn}}^{(\mathrm{new})} = \rho_{\mathrm{nn}}^{(\mathrm{old})} - \lambda_{\mathrm{lr}} \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\rho_{\mathrm{nn}}} \,. \tag{2.32}$$

An appropriate choice of the learning rate λ_{lr} is important. If the learning rate is too small, the training process will converge to a minimum very slowly, extending the training time. Conversely, minima may be skipped or never reached if the learning rate is too large. The learning rate is a parameter manually chosen before training and not determined by the optimization process. Parameters of this type are referred to as *hyperparameters*. Other examples of hyperparameters include the number of hidden layers and the batch size. The optimal hyperparameters are chosen through several training runs.

Batch normalization [IS15] is a common modification for neural networks that use mini-batches for training. It involves normalizing the inputs at each network layer for each mini-batch by subtracting the mean and dividing by the standard deviation. Additionally, two learnable parameters are introduced per batch norm, which optimally scale and shift the normalized values for that specific layer. As shown in [IS15], batch normalization can lead to a more stable training process and faster training through improved convergence. Therefore, it is also applied in the networks presented in this thesis.

2.4.2 | Convolutional Neural Networks

Simple neural networks such as MLPs, are suitable for illustrating the general concepts of deep learning. In practice, however, they are less suitable for many more complex problems. As an example in this thesis, neural networks are used to process input images. Consider a relatively small input image with 128×128 grayscale pixels; this results in $128^2 = 16\,384$ input neurons. Adding a hidden layer of the same size introduces $16\,384^2$ connections, resulting in more than 268 million weights. For context, AlexNet [KSH12], one of the pioneering networks that contributed to the success of convolutional layers in image processing networks, has only 60 million parameters. This network processes images with a resolution of 224×224 pixels and classifies them into 1000 different classes.

As mentioned earlier, networks with many parameters tend to overfit and require substantial training data. Furthermore, by considering each pixel individually, the MLP determines independent features for each pixel, ignoring spatial relationships crucial to understanding the image. Because of this feature independence, an image of the same object may be interpreted differently if the object is shifted by just one pixel. This missing attribute is called translation invariance.

Spatial understanding of structured data (e.g., images) can be improved by introducing *convolutional layers* to artificial neural networks. The convolutional layers of such a convolutional neural network (CNN) comprises *filters* that are applied across the image, incorporating neighboring pixel information. A filter is a matrix $\boldsymbol{\Gamma}$ with typically odd dimensions w_Γ and h_Γ. Mathematically, applying a filter $\boldsymbol{\Gamma}$ to an image $\boldsymbol{I}$ is a 2D convolution

$$(\boldsymbol{\Gamma} * \boldsymbol{I})(u,v) = \sum_{m=-(w_\Gamma-1)/2}^{(w_\Gamma-1)/2} \sum_{n=-(h_\Gamma-1)/2}^{(h_\Gamma-1)/2} \Gamma_{mn} \cdot \boldsymbol{I}(u-m, v-n)\,. \tag{2.33}$$

Convolutions cannot be applied to the edge pixels of the image, as image information is not available for all image elements necessary to calculate the filter result. There are several strategies to deal with this problem. For example, edge pixels can be ignored, resulting in an image with reduced dimensionality after applying the filter. Alternatively, edge pixels can be copied to the outside in a process called *padding*. The filter does not necessarily have to move pixel by pixel across the image. The *stride* indicates the number of pixels the filter moves at each step. It is noteworthy that a stride greater than 1 results in a reduction of the output dimensionality. For example, a stride of 2 effectively halves the resulting height and width.

In computer vision, different filters serve different purposes. For example, a Gaussian

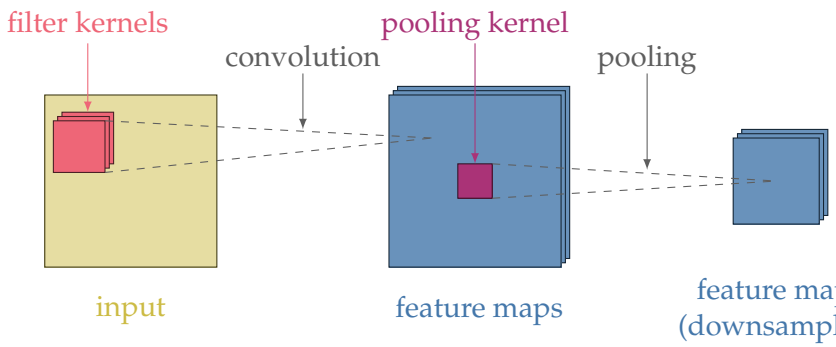


Figure 2.10: A visualization of a convolution layer with three filters and a pooling operation in CNN.

filter softens the image, while a Sobel filter is used for edge detection. In a CNN, the individual elements of the convolution matrix are not predefined but rather parameters determined during the training of the network. The network is optimized to shape the filters to independently extract the necessary features from the input images. The output of the convolution is called a feature map. The elements of the feature map are activated by a nonlinear activation function, typically a ReLU function. A *pooling* operation spatially reduces the activated feature maps. Pooling combines pixel values in a defined area (e.g., 2×2 pixels) of the feature map. Two common pooling operations are max pooling, which takes the maximum value of the area, and average pooling, which takes the average value of the area. Pooling is a nonlinear downsampling technique. It makes the network translationally invariant to a certain degree, reacting less sensitively to small changes in the positions of features. Additionally, downsampling reduces the feature maps' spatial dimensions—width and height—potentially reducing the complexity of subsequent connected layers.

In practice, a filter layer of the CNN may consist of several filters. For this purpose, the filter layer is extended by a third dimension by stacking several filters. The process of convolution and pooling is illustrated in Figure 2.10. Typically, CNNs comprise several convolution and pooling layers arranged sequentially, collectively referred to as the *feature extractor* of the network. For example, AlexNet [KSH12] has five convolution layers and three max pooling layers. In image classification, adding some fully connected layers to the feature extractor is common to base the classification on the previously determined features. The resulting fully connected classifier is much less complex, as it only processes the features derived from several pooling operations.

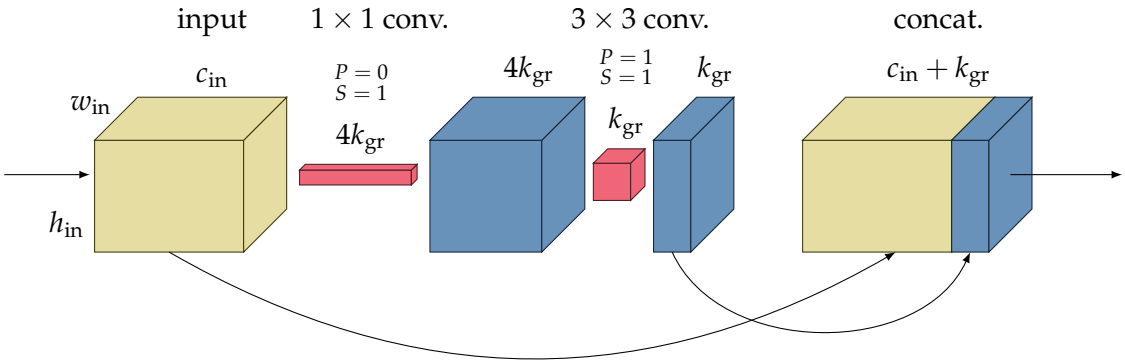
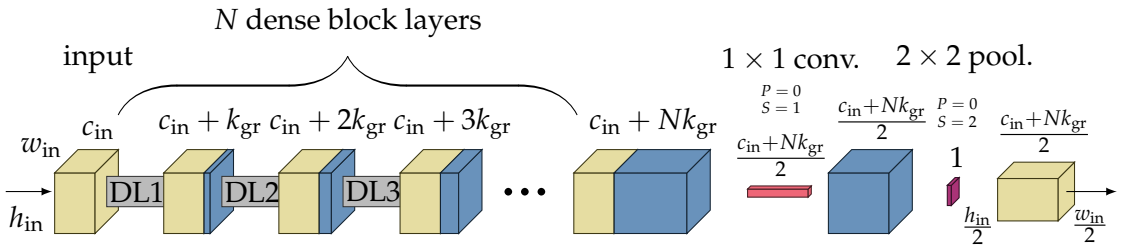


Figure 2.11: A schematic representation of a dense block layer with two convolutions and a concatenation.

2.4.3 | Densely Connected Convolutional Networks

An example of a modern CNN network architecture used in this thesis is the densely connected convolutional network (DenseNet) architecture [HLW17]. In the neural networks described before, the input of each new layer is the output of the previous layer, which is the product of a composite of operations. DenseNets, on the other hand, comprises dense blocks in which the input of a layer contains not only the output of the previous layer but also the concatenation of all the preceding layers in the block. The layers of a dense block are called dense block layers and are illustrated schematically in Figure 2.11. In this and the following schematic figures, the notation should be read as follows: the number above a block corresponds to the dimension of the channels or the number of feature maps (respectively, in the case of a filter layer, the number of filters). In general, the input of the dense block layer has dimensions $h_{\text{in}} \times w_{\text{in}} \times c_{\text{in}}$. Initially, a bottleneck comprising 1×1 convolutional filters is applied to the input, as suggested in [HLW17]. Deeper within the dense block, the number of channels c_{in} of a dense block layer has expanded through concatenations. The bottleneck operation reduces this dimensionality before the computationally intensive subsequent convolution, as proposed in [SVI+16]. The bottleneck results in a constant number of feature maps of spatial dimensions $h_{\text{in}} \times w_{\text{in}}$ to which k_{gr} additional 3×3 convolutional filters are applied. Here, k_{gr} represents the growth rate of the DenseNet and is a hyperparameter that describes how much the number of channels per dense block layer increases. It is important to note that because the feature maps within a dense block are concatenated, they must have the same height and width. Therefore, a padding of 1 (notated as $P = 1$ in Figure 2.11) must be applied to the 3×3 convolutional operation. The stride (notated as $S = 1$) is the same for both convolutional operations within the dense block layer. It is also worth mentioning that



(a) Dense block (b) Transition layer

Figure 2.12: A schematic representation of a dense block (a) comprising N dense block layers followed by a transition layer (b) that halves the dimensionality of the dense block.

batch normalization and a ReLU activation (recall Equation (2.26)) are applied before each convolutional operation, which is omitted in Figure 2.11 for readability. The result of the second convolution are k_{gr} feature maps with spatial dimensions $h_{\text{in}} \times w_{\text{in}}$, which are concatenated with the input, resulting in an $h_{\text{in}} \times w_{\text{in}} \times (c_{\text{in}} + k_{\text{gr}})$ dimensional output of the dense block layer.

A dense block comprises several successive dense block layers, as shown schematically in Figure 2.12a. The block configuration is a hyperparameter indicating the number of dense block layers within different dense blocks of the DenseNet. As previously mentioned, the width and height of the feature maps within a dense block layer remain constant throughout the entire dense block. However, as described in Section 2.4.2, an essential aspect of CNNs involves downsampling operations that progressively reduce the spatial dimensions of the feature maps as the network progresses.

Therefore, each dense block is succeeded by a transition layer (see Figure 2.12b), designed to achieve this downsampling. In the transition layer, batch normalization is applied first, followed by a ReLU activation (both omitted in Figure 2.12b for readability). Subsequently, the channel dimension is compressed by 1×1 convolutional filters, halving the number of feature maps. This compression is followed by downsampling by 2×2 average pooling with a stride of 2. The output of the transition layer is consequently halved in terms of height, width, and channels, serving as the input for the subsequent dense block.

Dense blocks and transition layers are the fundamental components of the DenseNet architecture. The exact composition depends, among other things, on the nature of the input data. To illustrate the network architecture in a concrete scenario, the design

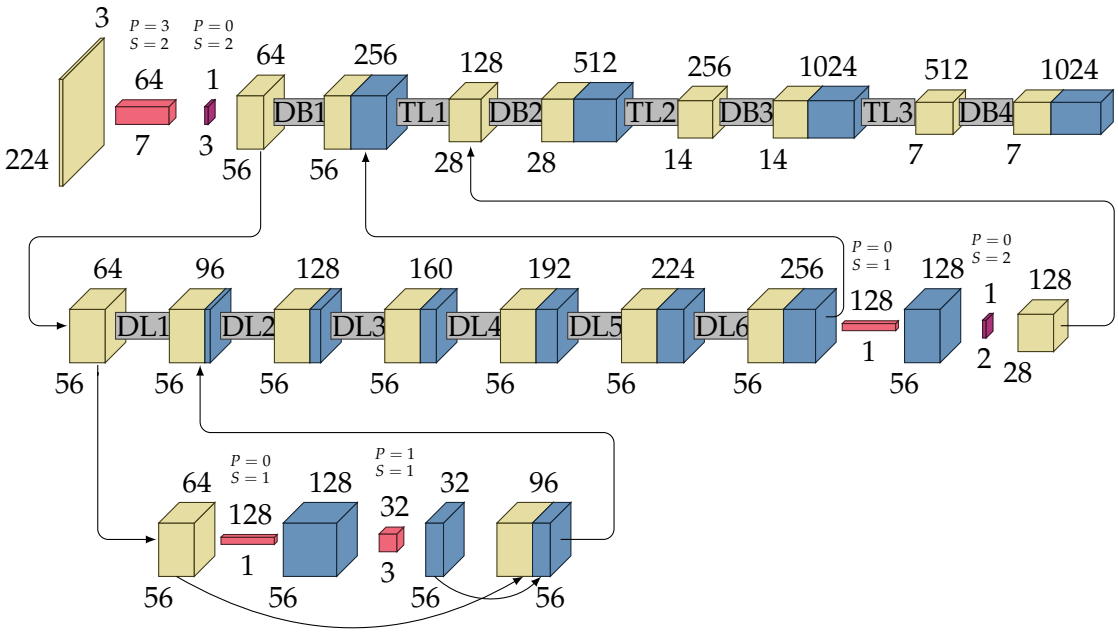


Figure 2.13: A schematic representation of the feature extractor of the used DenseNet architecture with the first dense block (DB1) and its transition layer (TL1) in more detail in the second row and the first dense block layer (DL1) of this block in the third row.

choices for training on the ImageNet [DDS+09] dataset are illustrated below. ImageNet is a large-scale image classification dataset comprising 1000 different classes. It is a widely used benchmark for image processing networks in deep learning. For networks designed to train on ImageNet, it is common to standardize the input images by resizing them to a square resolution of 224×224 pixels [KSH12]. An established DenseNet structure for ImageNet typically comprises four dense blocks with a block configuration of $[6, 12, 24, 16]$ [HLW17] (indicating the number of dense block layers per dense block). The corresponding feature extractor is illustrated schematically in Figure 2.13. The number below each element in this figure corresponds to both the height and width, given the square nature of the input image. Preceding the initial dense block, the input image passes through a 7×7 convolutional layer with 64 filters. Using a stride of 2 during this convolutional operation reduces the height and width of the input by half. Subsequently, a 3×3 max pooling operation, also with a stride of 2, further halves the spatial dimensions, resulting in $h_{\text{in}} = 56$ and $w_{\text{in}} = 56$ as the spatial dimensions for the input of the first dense block layer. After four dense blocks and three transition layers, the output of the feature extractor has a dimensionality of $7 \times 7 \times 1024$.

For image classification tasks, a classifier is subsequently applied to the features extracted by the feature extractor. A potential classifier for the proposed DenseNet architecture can be defined as follows: initially, a ReLU activation is applied to the output of the feature extractor, followed by a 7×7 average pooling operation. The resulting $1 \times 1 \times 1024$ dimensional latent vector is connected to a subsequent fully connected layer with 1000 neurons. Each output neuron corresponds to one of the 1000 classes in the ImageNet dataset. This output must then be normalized. For classification problems, normalization of the output is often achieved using the softmax activation function

$$\text{softmax}_i(\Psi^{(\text{est})}) = \frac{e^{\psi_i}}{\sum_j e^{\psi_j}}\,, \tag{2.34}$$

where $\Psi^{(\text{est})}$ represents the entire output, and ψ_i corresponds to the output for the i-th class. The softmax normalization scales all elements of the output to a range between 0 and 1 and constrains the sum of all elements to 1. Consequently, the normalized output can be interpreted as sort of "probability", indicating the likelihood that a given input image belongs to a specific class. For mathematical correctness, it is important to note that this is not a real probability, as the results depend not only on the input but also on the random initialization of the network parameters.

3

Sampling Techniques

The real environment exists in a continuous analog form, with several quantities such as geometry, texture, sound waves, and light varying smoothly across space and time. However, electronic devices (e.g., computers or MR devices) operate on discrete digital representations, storing and processing information in finite chunks. Sampling techniques refer to methods that approximate a continuous quantity as a discrete representation. The digital camera sensor, as described in Section 2.2, for example, samples the real environment at the grid points of its sensor pixels.

Sampling is, therefore, a fundamental operation in computer science and is used in many aspects of MR. For example, it plays an important role in the creation of virtual scenes. Virtual 3D objects are often represented as sampled surface meshes composed of flat polygons, which are a discrete approximation of the smoothly curved surfaces found in reality. The rendering algorithms for image synthesis, as described in Section 2.3, simulate the flow of light by evaluating the continuous rendering equation at discrete sample points. For path tracing, it is necessary to stochastically sample the surfaces of virtual 3D objects to generate light paths that achieve as noise-free rendering results as possible.

Sampling plays an important role not only in the visualization of virtual scenes but also in their animation. Real-time capable physically-based simulations often rely on particle representations of the objects to be simulated, including gases, fluids, granular materials, cloth, rigid bodies, and deformable objects. The particles serve as samples to discretize continuum properties such as velocity or density and to efficiently and accurately handle object and domain boundaries. The resolution of the sampling (i.e., the size of the particles) significantly impacts the computational cost and the level of detail of the simulation. Figure 3.1 illustrates the sampling of several objects within a particle-based simulation.

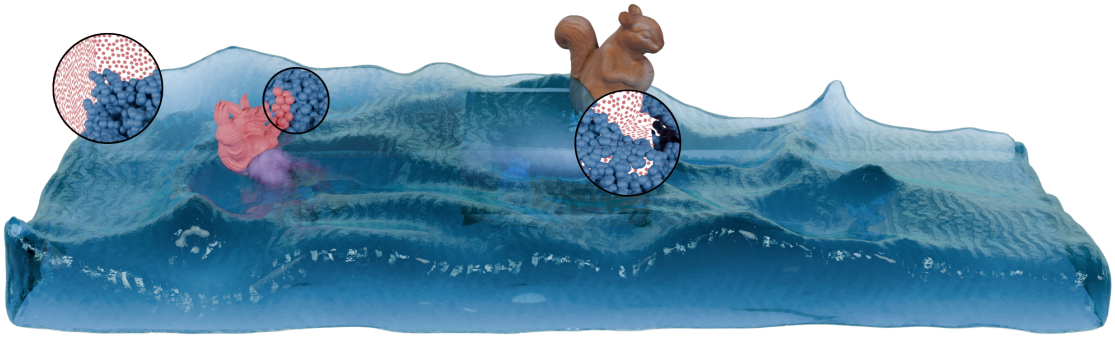


Figure 3.1: Simulation domain, rigid bodies and fluid sampled with particles and simulated in a unified simulation framework.

3.1 | Outline

This chapter presents fundamental methods for sampling the surfaces and volumes of virtual 3D objects. These methods were initially gathered for [SS21], where they were implemented in an application called LEAVEN[1], designed for sampling 3D objects in particle-based simulations. Although the application details of LEAVEN are not discussed in this thesis, the underlying sampling methods are broadly applicable to several topics covered in this work. Section 3.2 briefly introduces polygon meshes, which are the fundamental structure of the virtual 3D objects discussed in this chapter. Additionally, different characteristics that sampling can exhibit are described. Section 3.3 focuses on sampling surfaces, starting from a disk and extending to arbitrary 3D objects. Section 3.4 takes a step further, demonstrating how to sample the entire volume of 3D objects.

3.2 | Background

The sampling techniques described in this chapter are designed to generate sample points on the surface or within the volume of 3D objects. In computer graphics, 3D objects are composed of flat polygons with straight sides, known as *faces*. Each polygon comprises corner points—called *vertices*—connected by *edges*. The most common type of polygon for 3D meshes is the triangle. In this thesis, all the methods using 3D objects rely on triangle mesh representations. As already mentioned, these 3D meshes represent sampled representations of smoothly curved surfaces of real objects. This sampling

[1] LEAVEN is publicly available at: `https://github.com/a1ex90/Leaven`

process, referred to as *triangulation*, is not part of this work. For further information about polygon meshes and triangulation, refer to [BKP+10].

Depending on the intended use of the sample points, there are different requirements when generating samples on the surfaces or within the volumes of such 3D objects. These requirements are discussed in detail in the subsequent sections. In essence, the sampling techniques can be classified into different types of methods. In uniform sampling, the samples are distributed uniformly over the sampling domain without the domain's characteristics influencing the sampling result. In adaptive sampling, the sample point density adapts to the geometry of the sampling domain. For example, more samples are generated in regions with high curvature than in relatively flat regions. Additionally, there are methods where the sampling is non-deterministic and randomly distributed over the sampling domain. Nevertheless, it should be noted that such methods often rely on additional conditions that influence the sampling result. One such method is *Poisson disk sampling* [C86], which involves maintaining an additional minimum distance between individual samples to avoid clustering.

The irregularity pattern—also known as *noise*—is an important consideration for randomly distributed sample points. The examination of noise is most effective in the frequency domain, called the *Fourier spectrum*. A point distribution $\mathbb{S}$ with $|\mathbb{S}|$ samples can be transformed into the frequency domain using the *Fourier transformation*. Summing across all sample points $\boldsymbol{x}_j = (x_j, y_j, z_j)^\intercal \in \mathbb{S}$ yields the discrete Fourier coefficients

$$F_{\mathrm{FT}}(\nu_x, \nu_y, \nu_z) = k_{\mathrm{FT}} \sum_{j=1}^{|\mathbb{S}|} \mathrm{e}^{-2\pi i(\nu_x x_j + \nu_y y_j + \nu_z z_j)} \tag{3.1}$$

for the respective spatial frequencies ν_x, ν_y, and ν_z, where k_{FT} is a normalizing factor. The magnitude squared $|F_{\mathrm{FT}}|^2/|\mathbb{S}|$ is proportional to the power of the 3D sinusoidal oscillation with the frequencies ν_x, ν_y, and ν_z.

The amplitudes of this power spectrum provide information about different characteristics of the distribution. For example, if the sample points are evenly distributed in the spatial domain, strong peaks at regular intervals are visible in the spectrum. In general, different shapes of the power spectrum are classified as *colors of noise*. White noise is defined as a spectrum where all frequencies have equal amplitude. This phenomenon arises from the complete random distribution of the samples without any constraints, which may result in clusters or gaps in the point distribution.

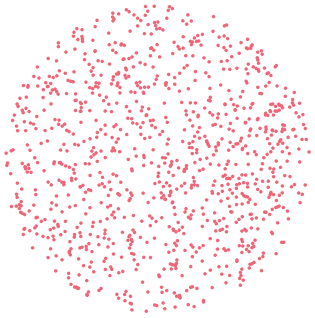
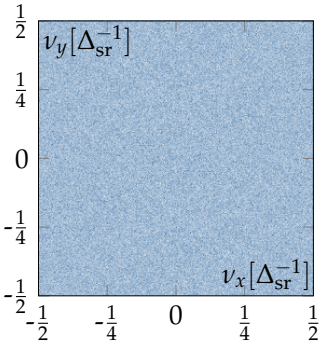
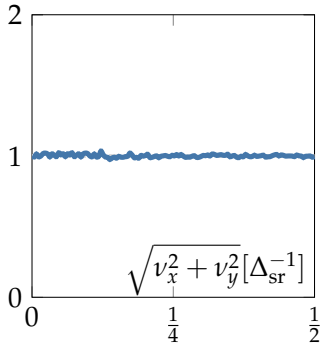
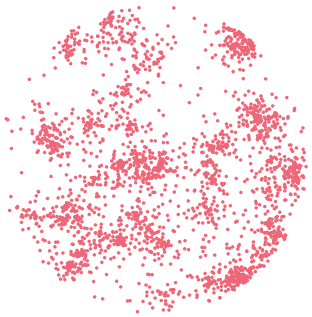
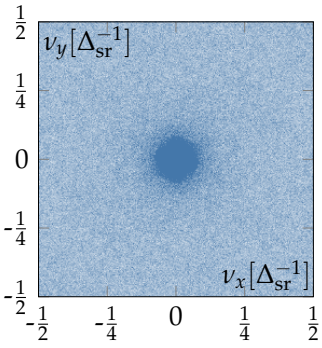
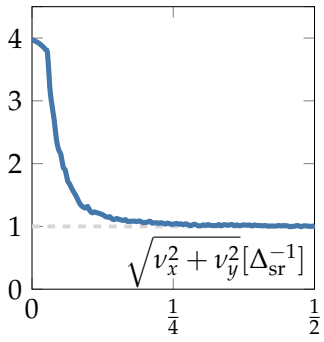
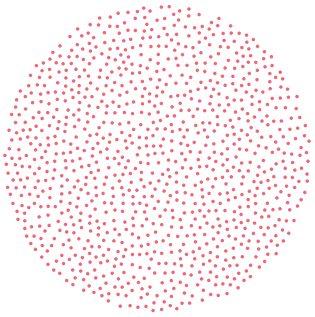
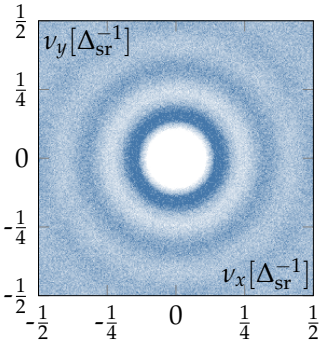
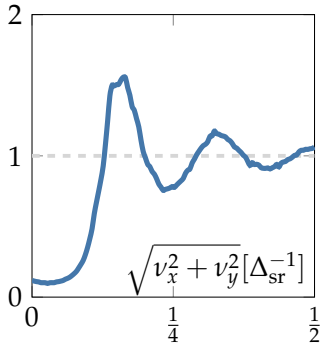
	Sampling	$\lvert F_{\mathrm{FT}}\rvert^2/\lvert \mathbb{S}\rvert$	Radial mean of $\lvert F_{\mathrm{FT}}\rvert^2/\lvert \mathbb{S}\rvert$
(a)			
(b)			
(c)			

Table 3.1: A disk sampled with 1000 points using different methods: (a) random locations resembling white noise, (b) clustered locations resembling brown noise, and (c) the method from Section 3.3.3 satisfying blue noise characteristics. For each sampling method, the normalized power spectrum $\lvert F_{\mathrm{FT}}\rvert^2/\lvert \mathbb{S}\rvert$ averaged over many runs is plotted using a sample rate of $\Delta_{\mathrm{sr}} = \frac{1}{256}$. Additionally, for visualization purposes, the radial mean of the normalized power spectrum is presented in a second plot.

In contrast, pink and brown noise exhibit strong amplitudes in the low-frequency range that decrease with increasing frequency. This indicates the presence of clusters or other large, correlated structures in the point distribution. These low frequencies are undesirable for most sampling purposes.

Conversely, at the opposite end of the spectrum is the *blue noise characteristic*, which is distinguished by an increase in amplitude with increasing frequency and a lack of low-frequency components. In point sampling, the definition of blue noise is often used less strictly. Point distributions commonly fulfill blue noise characteristics if they lack low-frequency energies in the power spectrum. They may display structured residual peaks and do not necessarily need to show increasing amplitude at higher frequencies. A point distribution with blue noise in the frequency domain is uniformly distributed in the spatial domain, lacks clusters or gaps, and is generally visually more appealing. Overall, blue noise is the optimal choice for many sampling tasks in computer graphics. In Table 3.1, samplings of a 2D disk with different noise colors are visualized, and the corresponding power spectrum is plotted. For a more comprehensive overview of noise in general, its colors, and its mathematical modeling, refer to [M19].

3.3 | Surface Sampling

Sampling surfaces of 3D objects is frequently required in several areas of computer graphics. In rendering, it is essential for tasks such as texture mapping, anti-aliasing, and, as demonstrated in Section 2.3.3, evaluating the rendering equation via path tracing. Achieving a uniform sampling distribution is often necessary to produce high-quality visual results. Sections 3.3.1 and 3.3.2 describe how to produce uniform samplings of spherical or disk-shaped surfaces, which are used in different examples in this thesis.

In particle-based simulations, surface sampling is important for defining boundary conditions for particle interactions with domain boundaries and static rigid bodies. The representation of objects or boundaries with sampled particles on their surfaces ensures that the simulated particles can perceive and interact with these objects, as described in [AIA+12]. Effective surface sampling in this context requires a balance: the sample density must be high enough to prevent particles from tunneling through boundaries, yet not so dense as to impact computational efficiency negatively. The sampling should be uniformly distributed to enable an even force distribution during collisions. However, it should also be avoided being arranged in regular patterns to prevent artifacts due to periodicity. Consequently, the sampling distribution should

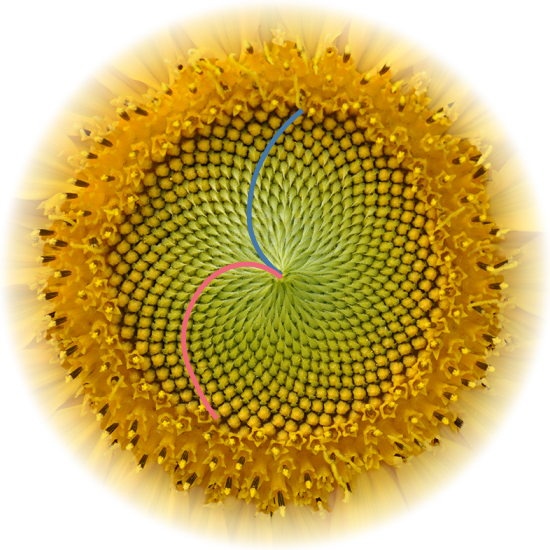


Figure 3.2: In the seed distribution pattern on the petal of a sunflower, two families of spiral arcs can always be observed, one with right-handed (clockwise) curvature (shown in blue) and one with left-handed (counter-clockwise) curvature (shown in red). [underlying sunflower image from yellow_sunflower_001/Wikimedia under CC license.]

exhibit blue noise characteristics. Such a method for the surfaces of arbitrary 3D objects is described in detail in Section 3.3.3.

3.3.1 | Uniform Disk Sampling

An elegant, nature-inspired method for uniform sampling of a disk can be found in the arrangement of seeds in a sunflower, known as the sunflower pattern. This naturally occurring arrangement arises from the sunflower's growth process and aims to distribute equal-sized seeds as densely packed as possible on the sunflower's disk-shaped petal. The pattern comprises two sets of spiral arcs, one running clockwise and the other counterclockwise (see Figure 3.2).

The entire pattern can also be mathematically described as a unique, tightly wound spiral, known as *generative spiral* [BB37]. The angular spacing of successive elements on this generative spiral is that of the golden angle

$$\Phi_\gamma = 2\pi \left(1 - \Phi_{\mathrm{gr}}^{-1}\right) \approx 137.5^\circ\,, \tag{3.2}$$

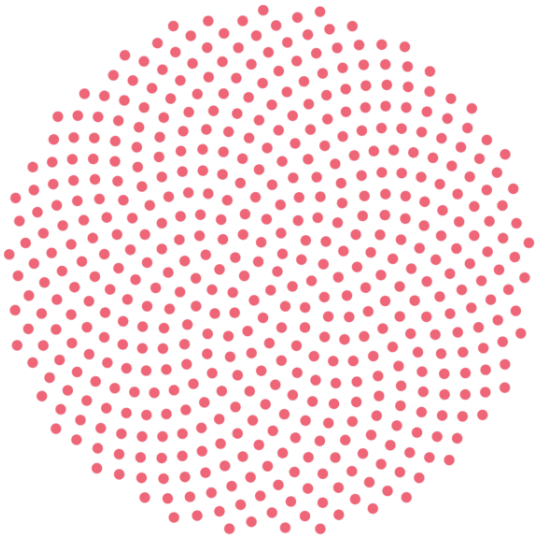


Figure 3.3: A disk uniformly sampled with $|\mathbb{S}_{\text{dsk}}| = 600$ points, where the points are placed at angular intervals of the golden angle Φ_γ on a Fermat spiral.

which is based on the golden ratio

$$\Phi_{\text{gr}} = \frac{\left(1+\sqrt{5}\right)}{2}. \tag{3.3}$$

This angular spacing ensures that in the sunflower, no two seeds are exactly aligned along the same radial line, resulting in a uniform and evenly distributed arrangement of seeds. For a mathematically ideal uniform distribution, the radius of the generative spiral must increase proportionally to the square root of the spiral angle [V79]. This form of the generative spiral is also known as the *Fermat spiral*. The Fermat spiral can be expressed as a set of polar coordinates (r, ϕ). This allows $|\mathbb{S}_{\text{dsk}}|$ samples with polar coordinates

$$r_i = \frac{r_{\text{dsk}}}{\sqrt{|\mathbb{S}_{\text{dsk}}| - \frac{1}{2}}} \sqrt{i - \frac{1}{2}}, \tag{3.4}$$

$$\phi_i = \Phi_\gamma \, i \tag{3.5}$$

to be defined on a disk with radius r_{dsk} for $i \in \{1, \ldots, |\mathbb{S}_{\text{dsk}}|\}$ (see Appendix A.1 for a detailed derivation). The offset of $1/2$ in Equation (3.4) serves to achieve the optimal uniform distribution in the vicinity of the disk's center, as demonstrated in [SJ06]. Figure 3.3 shows the result of such sampling of the surface of a disk.

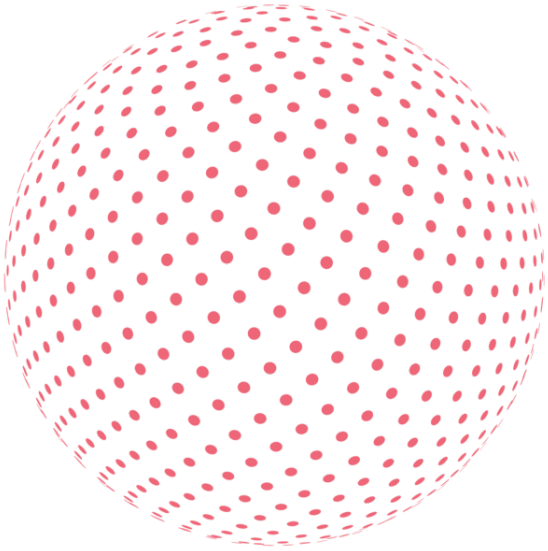


Figure 3.4: A sphere uniformly sampled with $|\mathbb{S}_{\text{sph}}| = 1000$ points, where the points are wrapped around the sphere as a projection of the Fermat spiral with angular intervals of the golden angle Φ_γ.

3.3.2 | Uniform Sphere Sampling

The previously described method for the uniform sampling of a disk can be modified to uniformly sample a sphere, as described in [SJ06]. This requires wrapping the planar grid around a sphere. In the 2D case, an angular spacing of Φ_γ was maintained on the Fermat spiral using polar coordinates. In the 3D case, spherical coordinates (recall Equations (2.4) - (2.6)) are suitable for describing the problem and exploiting the underlying radial symmetry. To achieve a uniform sampling on the sphere's surface with a constant radius r_{sph}, the distribution of the sample points must be appropriately spaced in both the azimuthal ϕ and polar angle θ. The distribution of the azimuthal angles ϕ_i follows the same logic as in the 2D case in Equation (3.5). To achieve uniform spacing in the up direction, it is necessary to ensure that each slice of the sphere has the same density of points. The radius of each circular slice of the sphere is proportional to the cosine of the polar angle θ. Consequently, to achieve an even distribution, the cosine of the polar angle $\theta \in [0, \pi]$ must be sampled linearly over its entire range $-1 \leq \cos\theta \leq 1$. Assuming that the number of sample points $|\mathbb{S}_{\text{sph}}|$ is even, each hemisphere is sampled with $|\mathbb{S}_{\text{sph}}|/2$ points, which for a range of $j \in \{-|\mathbb{S}_{\text{sph}}|/2, \ldots, |\mathbb{S}_{\text{sph}}|/2\}$ leads to:

$$\cos\theta_j = -\frac{j}{|\mathbb{S}_{\text{sph}}|/2}\,. \tag{3.6}$$

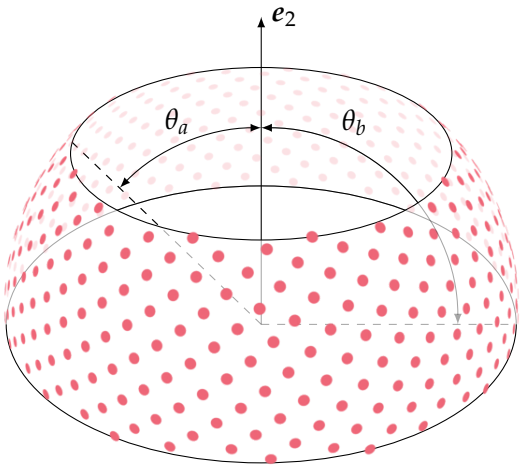


Figure 3.5: A ring-shaped area on a spherical surface between two polar angles θ_a and θ_b uniformly sampled with $|\mathbb{S}_{\text{rng}}| = 800$ points.

This can be transformed into the range $i \in \{1, \ldots, |\mathbb{S}_{\text{sph}}|\}$ for arbitrary numbers of $|\mathbb{S}_{\text{sph}}|$. Ultimately, a sample on the sphere's surface can be described by a set of spherical coordinates

$$r_i = r_{\text{sph}}\,, \tag{3.7}$$

$$\theta_i = \arccos\left(\frac{(|\mathbb{S}_{\text{sph}}| + 1) - 2i}{|\mathbb{S}_{\text{sph}}|}\right), \tag{3.8}$$

$$\phi_i = \Phi_\gamma\, i\,. \tag{3.9}$$

Here, an offset is also used, motivated by the same reasons as previously mentioned in disk sampling. Figure 3.4 shows the result of such uniform sampling of the surface of a sphere.

Often, it is desired to sample only a portion of a sphere. A typical example is the upper hemisphere in rendering. Such sampling can be achieved with the previously described method by adjusting the range in i. Samples exclusively on the upper hemisphere are obtained by setting the range to $i \in \{1, \ldots, \lfloor(|\mathbb{S}_{\text{sph}}| + 1)/2\rfloor\}$. More generally, the sampling area can be constrained to:

$$i \in \left\{\left\lceil\frac{|\mathbb{S}_{\text{sph}}|(1-\cos\theta_a)+1}{2}\right\rceil, \ldots, \left\lfloor\frac{|\mathbb{S}_{\text{sph}}|(1-\cos\theta_b)+1}{2}\right\rfloor\right\}, \tag{3.10}$$

between a minimum polar angle θ_a and a maximum polar angle θ_b, where $0 \leq \theta_a < \theta_b < \pi$. If a specific number $|\mathbb{S}_{\text{rng}}|$ of samples is required in this area, the total number of

samples for the sphere

$$|\mathbb{S}_{\text{sph}}| = \left\lceil \frac{2|\mathbb{S}_{\text{rng}}|}{\cos\theta_a - \cos\theta_b} \right\rceil \tag{3.11}$$

can be adjusted accordingly. Figure 3.5 illustrates the sampling of a ring-shaped area on the sphere between two specified polar angles.

3.3.3 | Uniform Sampling of Arbitrary Objects

The preceding surface sampling techniques describe samples on algebraic surfaces. For arbitrary 3D objects with a volume V_{obj}, an algebraic description of the surface ∂V_{obj} generally does not exist. To describe arbitrary objects in computer graphics, triangle meshes (recall Section 3.2) are an appropriate choice. Surface sampling of arbitrary objects aims to obtain a set of sample points $\mathbb{S}_{\text{obj}}$ on the faces of these meshes.

One popular method for surface sampling is Poisson disk sampling [C86]. It is a randomly distributed sampling method in which all sample points maintain a minimum distance l_{dst} from each other. This implies

$$\text{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq l_{\text{dst}} \tag{3.12}$$

for any arbitrary pair of sample points $\boldsymbol{x}_i$, $\boldsymbol{x}_j \in \mathbb{S}_{\text{obj}}$. There are different metrics for evaluating the distance between two points. The most intuitive is the Euclidean distance

$$\text{dist}_E(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}\,, \tag{3.13}$$

which describes the straight-line distance between the two points. Another norm useful for surface sampling is the Geodesic distance

$$\text{dist}_G(\boldsymbol{x}_i, \boldsymbol{x}_j) = \inf\left\{ \int \gamma_{\text{pth}}(s)\,\text{d}s \;\middle|\; \gamma_{\text{pth}} \in \Gamma_{\partial V_{\text{obj}}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \right\}, \tag{3.14}$$

which describes the length of the shortest path γ_{pth} of all possible paths $\Gamma_{\partial V_{\text{obj}}}$ between the two points along the surface. The exact calculation of geodesic distance on polygon meshes is demonstrated in [MMP87], and in [SSK+05], the efficient approximation of this distance.

Using Poisson disk sampling, uniformly distributed and non-clustered point distributions in the spatial domain can be formed without regular patterns, as required at the beginning of this section. A brute force method for Poisson disk sampling relies on dart throwing, as suggested in [C86]. This approach repeatedly chooses random points on the surface, analogous to a bad dart player hitting random spots on a dartboard. To

determine a random point uniformly on the surface of polygon meshes, a random face must be chosen. This selection must be weighted by the area of the faces to ensure equal probability for each point on the surface. For a face with an index j and an area A_{fc_j}, this corresponds to a probability

$$P_j = \frac{A_{\mathrm{fc}_j}}{A_{\mathrm{obj}}} \tag{3.15}$$

with an area

$$A_{\mathrm{obj}} = \sum_i A_{\mathrm{fc}_i} \tag{3.16}$$

for the entire polygon mesh. In the case of triangle meshes, a face comprises vertices $\boldsymbol{v}_a$, $\boldsymbol{v}_b$, and $\boldsymbol{v}_c$ and has an area

$$A_{\mathrm{fc}} = \frac{1}{2} \left| (\boldsymbol{v}_b - \boldsymbol{v}_a) \times (\boldsymbol{v}_c - \boldsymbol{v}_a) \right| . \tag{3.17}$$

A random point

$$\boldsymbol{x}_i = \lambda_u \boldsymbol{v}_a + \lambda_v \boldsymbol{v}_b + \lambda_w \boldsymbol{v}_c \tag{3.18}$$

on a face can be defined using barycentric coordinates

$$\lambda_u = 1 - \sqrt{\rho_{\mathrm{rnd}_1}} , \tag{3.19}$$

$$\lambda_v = \rho_{\mathrm{rnd}_2} \sqrt{\rho_{\mathrm{rnd}_1}} , \tag{3.20}$$

$$\lambda_w = 1 - \lambda_u - \lambda_v \tag{3.21}$$

with two uniformly distributed random values ρ_{rnd_1} and $\rho_{\mathrm{rnd}_2} \in [0, 1]$. If a randomly chosen point $\boldsymbol{x}_i$ does not violate the distance criterion (Equation (3.12)) with any existing points $\boldsymbol{x}_j$ in the sample set, it is added to the sample set. As the number of sample points increases, the resulting distribution more closely approximates blue noise characteristics [LD08]. However, as the sampling density increases, the number of attempts to fill the remaining gaps on the surface also increases. Further, it should be noted that algorithmic convergence is not guaranteed with this brute-force attempt.

A more elegant method with guaranteed convergence in linear runtime concerning the number of sample points is described in [B07]. Starting with an initial sample point $\boldsymbol{x}_0$, randomly chosen on the surface, N_t random candidate points are chosen within a radius between l_{dst} and $2l_{\mathrm{dst}}$ around $\boldsymbol{x}_0$ on the surface. For a candidate point $\boldsymbol{x}_{\mathrm{cp}}$ to qualify as a sample added to the sample set $\mathbb{S}_{\mathrm{obj}}$, it must not violate the distance criterion (Equation (3.12)) with respect to any already existing sample points $\boldsymbol{x}_j \in \mathbb{S}_{\mathrm{obj}}$. In the beginning, $\mathbb{S}_{\mathrm{obj}}$ is only populated with the initial sample point $\boldsymbol{x}_0$. For each new sample point $\boldsymbol{x}_i$ defined in this way, another N_t random candidate points are chosen around $\boldsymbol{x}_i$ on the surface.

The algorithm terminates when none of the remaining candidate points is an eligible new sample.

Comparing a candidate point with each sample point would exponentially increase the runtime of the algorithm with each additional sample. Therefore, it is advisable to introduce a search structure to reduce the number of collision detections. To limit the search area, the external dimensions of the object—the *bounding box* $\mathcal{B}_o$—should be determined. The bounding box is defined as an axis-aligned cube spanned by a minimum and maximum point. The minimum point

$$\boldsymbol{b}_{o_{\min}} = \left(\min_{\text{verts}\, i}(v_{i_x}),\ \min_{\text{verts}\, i}(v_{i_y}),\ \min_{\text{verts}\, i}(v_{i_z}) \right)^{\mathsf{T}} \tag{3.22}$$

is defined as the component-wise minimum of the respective three coordinate axes of all vertices. The maximum point

$$\boldsymbol{b}_{o_{\max}} = \left(\max_{\text{verts}\, i}(v_{i_x}),\ \max_{\text{verts}\, i}(v_{i_y}),\ \max_{\text{verts}\, i}(v_{i_z}) \right)^{\mathsf{T}} \tag{3.23}$$

is defined as the component-wise maximum. The two points define the size

$$b_{o_k} = b_{o_{\max_k}} - b_{o_{\min_k}} \quad \text{for } k \in \{x, y, z\} \tag{3.24}$$

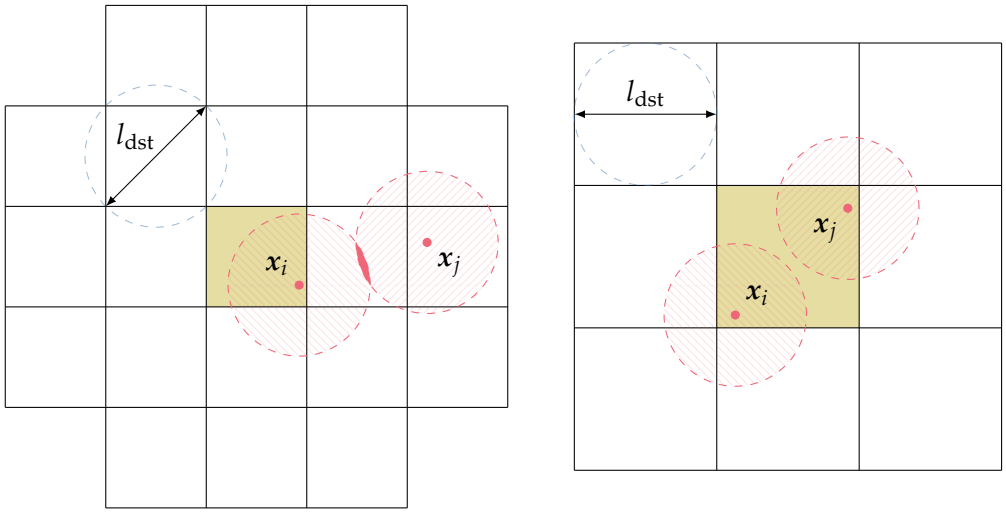
of the bounding box in each coordinate axis.

The bounding box $\mathcal{B}_o$ can be subdivided into a 3D grid to create an appropriate search structure for the sampling domain. In [B07], the grid cell size is chosen such that the diagonal of the cell corresponds to the minimum distance l_{dst}, ensuring that only a single sample point can be placed within a cell. However, this approach has a significant drawback: in 2D, 17 neighboring cells must be checked (see Figure 3.6a), and in 3D, already 80 cells must be checked. Alternatively, if the cell edge length is set to the minimum distance l_{dst}, multiple sample points can occupy a single cell (see Figure 3.6b), but only the cell itself and its 26 neighbors in 3D need to be checked. Consequently, the 3D grid of the bounding box contains

$$\left[\left\lceil \frac{b_{o_x}}{l_{\text{dst}}} \right\rceil, \left\lceil \frac{b_{o_y}}{l_{\text{dst}}} \right\rceil, \left\lceil \frac{b_{o_z}}{l_{\text{dst}}} \right\rceil \right] \tag{3.25}$$

cells. For a given candidate point $\boldsymbol{x}_{\text{cp}} = (x_{\text{cp}}, y_{\text{cp}}, z_{\text{cp}})^{\mathsf{T}}$, the indices

$$\left[\left\lfloor \frac{x_{\text{cp}} - b_{o_{\min_x}}}{l_{\text{dst}}} \right\rfloor, \left\lfloor \frac{y_{\text{cp}} - b_{o_{\min_y}}}{l_{\text{dst}}} \right\rfloor, \left\lfloor \frac{z_{\text{cp}} - b_{o_{\min_z}}}{l_{\text{dst}}} \right\rfloor \right] \tag{3.26}$$



(a) (b)

Figure 3.6: In (a), the cell diagonal equals the minimum distance l_{dst} between two sample points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, allowing only one sample point per cell, but requiring 17 neighboring cells to be checked. In (b), the cell edge length equals the minimum distance, allowing two sample points per cell, but only eight neighboring cells need to be checked.

of the corresponding grid cell can be determined. To validate a candidate point $\boldsymbol{x}_{\text{cp}}$ as a new sample, it is sufficient to check whether its grid cell or any of the 26 neighboring cells contain a sample point $\boldsymbol{x}_j$ that violates the distance criterion (Equation (3.12)) relative to the candidate point.

When using 3D objects in the form of polygon meshes (e.g., triangle meshes), choosing random candidate points distributed uniformly at a distance between l_{dst} and $2l_{\text{dst}}$ around a sample point on the surface presents a challenge. The potential region for candidate points may extend across multiple faces. Often, only a portion of a specific face falls within this potential region, complicating the calculation of the probability with which each face should be selected. In [BWW+10], it is suggested to initially define a significantly large set of random candidate points $\mathbb{S}_{\text{cp}}$ uniformly distributed across the entire surface. Rather than choosing new random candidate points in the vicinity of a sample point $\boldsymbol{x}_i$, candidate points $\boldsymbol{x}_{\text{cp}} \in \mathbb{S}_{\text{cp}}$ of the neighboring grid cells of $\boldsymbol{x}_i$ are considered. Given that the candidate set is uniformly distributed over the surface and is sufficiently large in relation to the surface, no additional bias is introduced. For the examples in this thesis, a size of $|\mathbb{S}_{\text{cp}}| = N_t A_{\text{obj}} / (\pi l_{\text{dst}}^2)$ is used for the candidate set.

This modification requires two grid structures, each with identical dimensions. The first

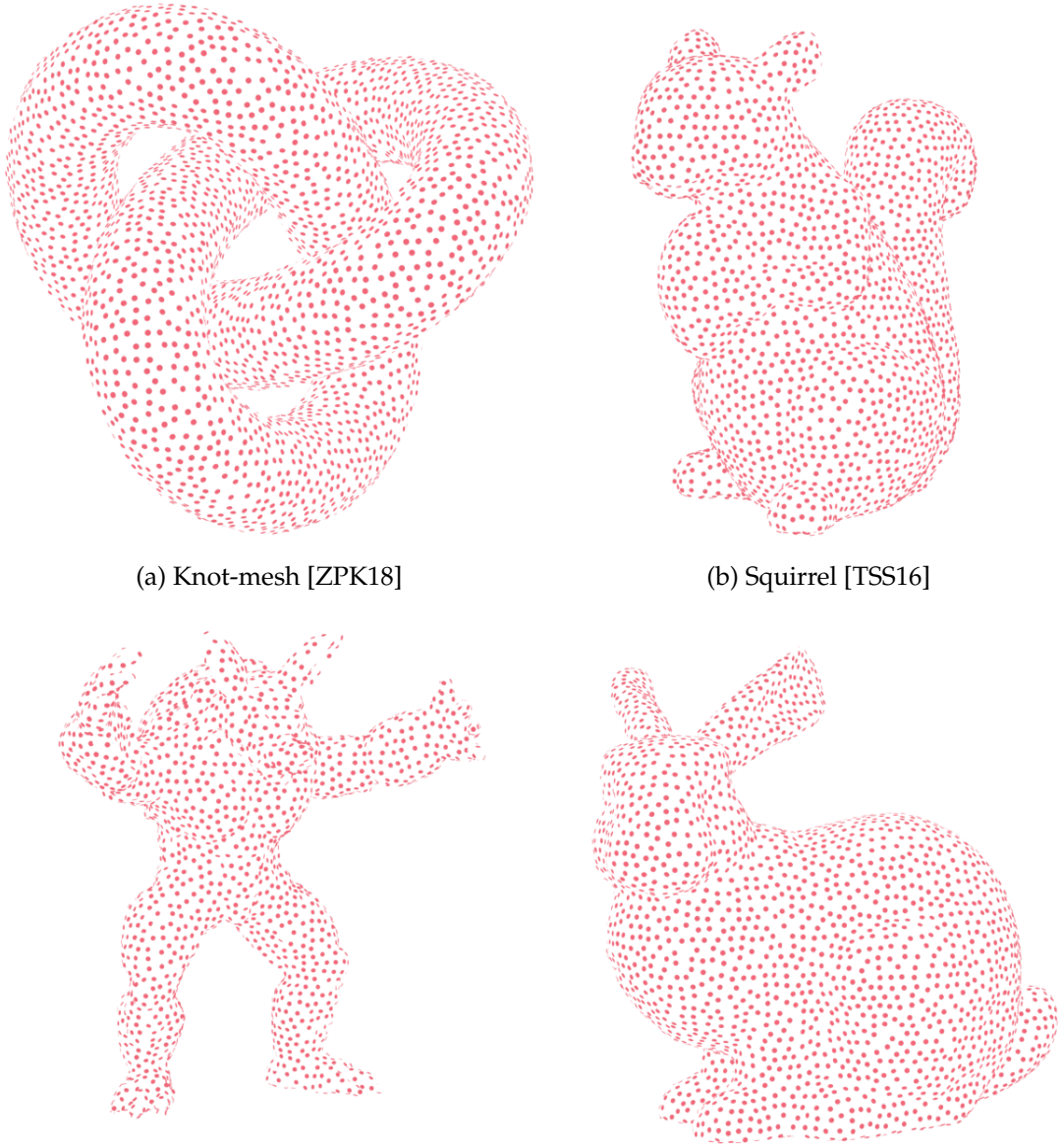
one is the candidate grid containing all candidate points $\boldsymbol{x}_{\text{cp}} \in \mathbb{S}_{\text{cp}}$. The second is the sample grid, which is initially empty and into which validated sample points $\boldsymbol{x}_i \in \mathbb{S}_{\text{obj}}$ are subsequently inserted. Furthermore, an active list is maintained that contains all sample points $\boldsymbol{x}_i \in \mathbb{S}_{\text{obj}}$ in whose neighborhoods potential new sample points $\boldsymbol{x}_{\text{cp}} \in \mathbb{S}_{\text{cp}}$ may still exist that do not violate the distance criterion (Equation (3.12)) to any existing sample points $\boldsymbol{x}_j \in \mathbb{S}_{\text{obj}}$.

Algorithm 1: Surface sampling of arbitrary 3D objects

Data: 3D mesh, $|\mathbb{S}_{\text{cp}}|$, l_{dst}
Result: $\mathbb{S}_{\text{obj}}$

$\mathbb{S}_{\text{cp}} \leftarrow |\mathbb{S}_{\text{cp}}|$ random points on the object's surface (faces of the mesh)
populate candidate grid `c_grid` with all $\boldsymbol{x}_{\text{cp}} \in \mathbb{S}_{\text{cp}}$
initialize empty sample grid `s_grid` and empty active list `a_list`
pick initial sample $\boldsymbol{x}_0 \in \mathbb{S}_{\text{cp}}$
place $\boldsymbol{x}_0$ in `s_grid` and add to $\mathbb{S}_{\text{obj}}$ and `a_list`
while `a_list` not empty **do**
    `found` $\leftarrow$ **false**
    pick random $\boldsymbol{x}_i$ from `a_list`
    forall $\boldsymbol{x}_{\text{cp}}$ in neighboring cells to $\boldsymbol{x}_a$ in `c_grid` **do**
        /* check if $\boldsymbol{x}_{\text{cp}}$ has no distance conflicts in $\mathbb{S}_{\text{obj}}$ */
        forall $\boldsymbol{x}_j$ in cell of $\boldsymbol{x}_{\text{cp}}$ and in neighboring cells to $\boldsymbol{x}_{\text{cp}}$ in `s_grid` **do**
            if $\text{dist}(\boldsymbol{x}_{\text{cp}}, \boldsymbol{x}_j) < l_{\text{dst}}$ **then**
                continue to next $\boldsymbol{x}_{\text{cp}}$
        /* $\boldsymbol{x}_{\text{cp}}$ is a valid new sample */
        `found` $\leftarrow$ **true**
        place $\boldsymbol{x}_{\text{cp}}$ in `s_grid`, `a_list`, and $\mathbb{S}_{\text{obj}}$
        break
    if not `found` **then**
        remove $\boldsymbol{x}_i$ from `a_list`
return $\mathbb{S}_{\text{obj}}$

The complete algorithm is presented in Algorithm 1 in pseudocode. After $|\mathbb{S}_{\text{cp}}|$ candidate points have been defined and inserted into the candidate grid, one of these candidates is selected as the initial sample point $\boldsymbol{x}_0$ and added to the active list. As long as the active list remains non-empty, a random sample point $\boldsymbol{x}_i$ from this list is selected. In the initial iteration, this is necessarily the initial sample. Subsequently, all candidate points $\boldsymbol{x}_{\text{cp}} \in \mathbb{S}_{\text{cp}}$ from the vicinity of the active sample point $\boldsymbol{x}_i$ are considered potential new samples. Therefore, candidates are gathered from the candidate grid cell corresponding



(a) Knot-mesh [ZPK18]

(b) Squirrel [TSS16]

(c) Armadillo [KL96]

(d) Bunny [TL94]

Figure 3.7: Random sample points are distributed uniformly across the surface of 3D models: In (a) with 6302 sample points, in (b) with 3950, in (c) with 3182, and in (d) with 4589. The size of the points is smaller than the minimum distance l_{dst} between samples for visualization purposes.

to the cell index of the sample point $\boldsymbol{x}_i$ (see Equation (3.26)) and from the 26 neighboring candidate grid cells. For each of these candidate points $\boldsymbol{x}_{\text{cp}}$, it is tested whether it violates the distance criterion with respect to existing sample points $\boldsymbol{x}_j \in \mathbb{S}_{\text{obj}}$. For this verification, only the 26 neighboring sample grid cells to the candidate's cell index, as well as its sample grid cell itself, need to be considered. If an existing sample point violates the distance criterion, the candidate point in question $\boldsymbol{x}_{\text{cp}}$ is discarded, and the subsequent candidate point is evaluated. If the distance criterion is not violated, the candidate becomes a valid new sample and is added to the sample set and grid. Further, it is included in the active list to identify potential new samples in its vicinity. If no candidate point $\boldsymbol{x}_{\text{cp}} \in \mathbb{S}_{\text{cp}}$ is identified as a valid new sample for a given sample point $\boldsymbol{x}_i$ from the active list, this sample point $\boldsymbol{x}_i$ is removed from the list. Each iteration either produces a new sample and adds it to the active list or removes a sample from the active list. Therefore, exactly $2|\mathbb{S}_{\text{obj}}| - 1$ iterations are needed to create $|\mathbb{S}_{\text{obj}}|$ sample points.

Figure 3.7 shows four widely-used 3D models and their surface sampling created using this method. The method can be parallelized, as shown in [BWW+10], which is beneficial for time-sensitive applications where dynamically changing geometries need to be sampled.

3.4 | Volume Sampling

Not only the sampling of surfaces of 3D objects, as demonstrated before, but also the sampling of their entire volumes is frequently used in computer graphics. In rendering, for example, volume sampling is used to visualize volumetric effects within a scene. Translucent materials, where light penetrates the surface, scatters within it, and then exits, require volume sampling to calculate light scattering accurately. These materials are simulated through a technique called *subsurface scattering* [JML+01]. It can realistically describe the optical behavior of substances ranging from liquid emulsions (e.g., milk) [FCJ07], to organic materials (e.g., skin) [KB04], and even rigid materials (e.g., marble) [JB02]. Additionally, clouds can be realistically visualized by sampling the volume of the cloud form to express the density and distribution of water droplets within the cloud [NDN96]. This allows simulation of the complex light interactions that create the characteristic appearance of clouds, including shadows within clouds, soft edges, and glowing rims.

Volume sampling also plays a significant role in particle-based simulations. It is necessary to sample the volume in which the medium is located in the initial state of the simulation

to establish the initial positions of particles, which may represent elements, such as fluids, gases, or granular materials. If particles are arranged in regular patterns, the simulation may initially be in equilibrium, resulting in no movement without external influences. A more realistic approach is to introduce a certain degree of imbalance at the start of the simulation, resulting in more dynamic and visually appealing results. This imbalance can be achieved by randomly sampling the volume with a uniform distribution. It is important to ensure that the particles do not overlap initially, as this may introduce ghost forces to the system. This can be realized by enforcing a minimum distance between samples, as in the surface sampling method in Section 3.3.3. In a unified particle simulation, as in [MMC+14], the interactions with rigid bodies and their motion can also be simulated alongside different media or deformable objects. Section 3.3 already discussed static rigid bodies and domain boundaries and how surface sampling can make them visible for particle simulation. These static surface particles are treated as infinite mass particles. For dynamic rigid bodies, each sampled particle has a mass corresponding to the volume it occupies and the object's material density. Here, it is more appropriate to sample the entire volume of the object. When the sampling is evenly arranged, each particle can be assigned the same mass.

The sample points required for rendering do not necessarily possess spatial extent, whereas particles typically have a spherical volume. It is crucial to define whether the sample point (i.e., the center of the particle) must be located in the volume or if the entire particle must be contained in it. For the methods discussed in the following, the convention is that only the center of the particle needs to be within the volume. This approach is consistent with the surface sampling methods discussed in the previous section, where the centers are located on the surface. Consequently, these methods can be applied to both rendering and simulation tasks. Appropriate handling must then be considered during simulation to account for this convention.

3.4.1 | Uniform Sampling of Arbitrary Objects

Section 3.3.3 described sampling the surface ∂V_{obj} of an arbitrary 3D object with a volume V_{obj}. This method recursively chooses samples that maintain a minimum distance from each other by evaluating randomly distributed candidates in the vicinity of existing samples. Defining these candidate points requires knowing how to create points that lie on the surface. To obtain suitable candidate points for volume sampling, defining points within the volume is necessary. Therefore, it is necessary to ensure that the volume, again represented by a triangle mesh, has a well-defined interior and exterior. This requires the

mesh to have a coherent topology, avoiding ambiguities caused by holes, openings, or inconsistent face orientations.

Whether a point $\boldsymbol{x}$ is inside or outside a closed polygon mesh can be determined using the ray casting algorithm [S62]. Starting from $\boldsymbol{x}$, a ray is cast in an arbitrary direction. This ray is tested for intersections with each face of the polygon mesh. If the ray intersects an even number of faces, the point $\boldsymbol{x}$ is outside the mesh. Conversely, if the ray intersects an odd number of faces, the point $\boldsymbol{x}$ is inside the mesh. This observation is mathematically supported by the Jordan curve theorem [C87]. Whether a ray intersects a triangular face can be tested using the Möller-Trumbore intersection algorithm [AT97]. The Möller-Trumbore algorithm fails when a ray intersects the edge or vertex of a triangle, which can be prevented by carefully handling edge and vertex cases through a modified edge-testing scheme [WBW13]. The number of faces that need to be tested for a given ray can be significantly reduced by using spatial acceleration structures such as bounding volume hierarchies [MOB+21], k-d trees [HH11], or octrees [WSC+95].

If the given mesh is not watertight, a winding number test [JKS13] can be used instead of ray casting, as it handles non-closed meshes better. This method calculates, for a point $\boldsymbol{x}$, how much of the solid angle

$$\Omega_f(\boldsymbol{x}) = 2\,\mathrm{atan}\left(\frac{\det\begin{bmatrix}\boldsymbol{e}_a & \boldsymbol{e}_b & \boldsymbol{e}_c\end{bmatrix}}{|\boldsymbol{e}_a|\cdot|\boldsymbol{e}_b|\cdot|\boldsymbol{e}_c| + (\boldsymbol{e}_a^\mathsf{T}\boldsymbol{e}_b)\boldsymbol{e}_c + (\boldsymbol{e}_a^\mathsf{T}\boldsymbol{e}_b)\boldsymbol{e}_c + (\boldsymbol{e}_a^\mathsf{T}\boldsymbol{e}_b)\boldsymbol{e}_c}\right) \tag{3.27}$$

is occupied by each triangular face with its vertices $\boldsymbol{v}_a$, $\boldsymbol{v}_b$, and $\boldsymbol{v}_c$ as seen from $\boldsymbol{x}$ [VS83]. Here, for better readability: $\boldsymbol{e}_a = \boldsymbol{v}_a - \boldsymbol{x}$, $\boldsymbol{e}_b = \boldsymbol{v}_b - \boldsymbol{x}$, and $\boldsymbol{e}_c = \boldsymbol{v}_c - \boldsymbol{x}$. The resulting winding number

$$W(\boldsymbol{x}) = \frac{1}{4\pi}\sum_f \Omega_f(\boldsymbol{x}) \tag{3.28}$$

is a continuous measure, in contrast to the binary result of the ray casting algorithm. W approaches 1 for points inside the mesh and 0 for points outside the mesh.

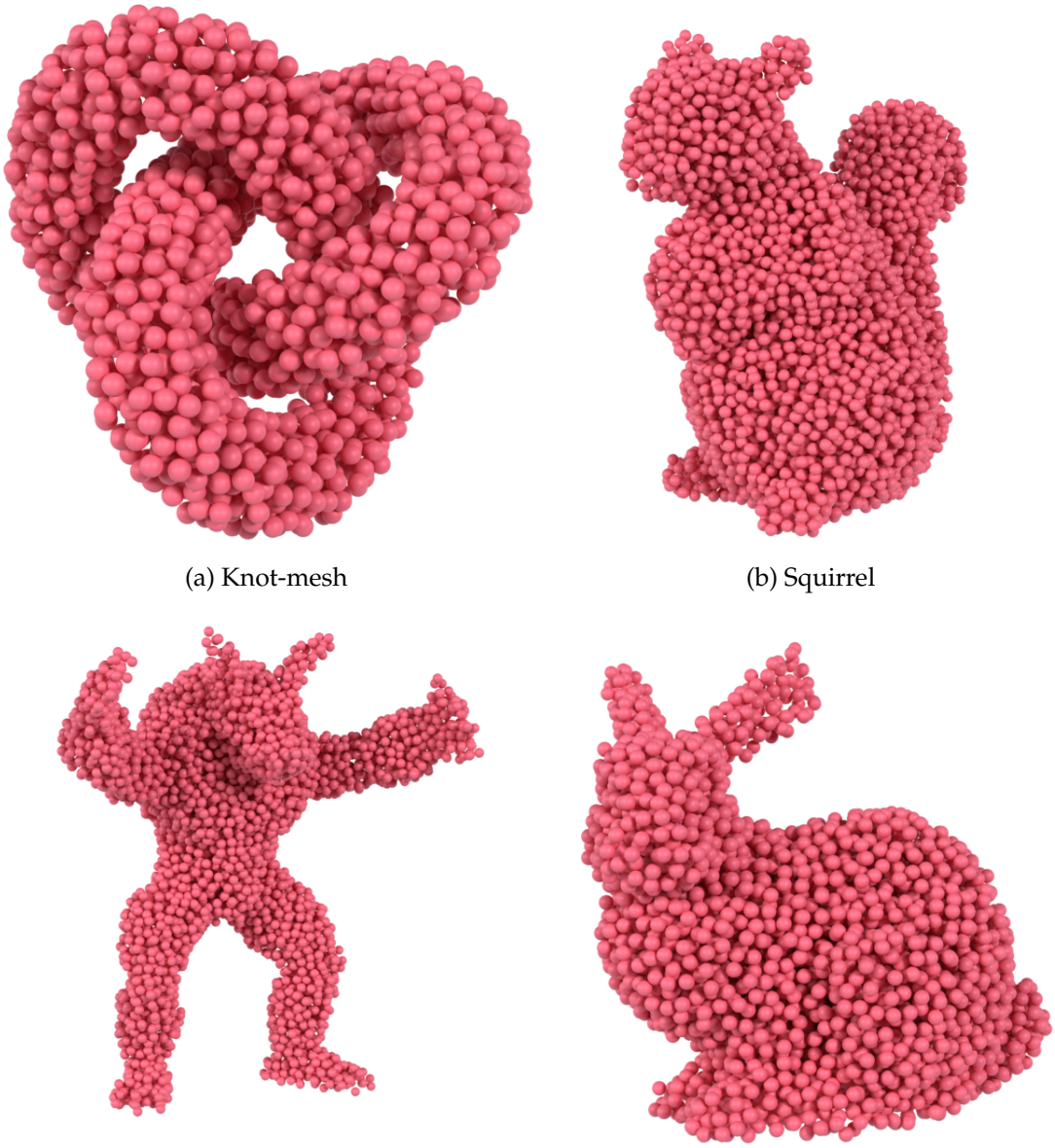
The ability to determine whether an arbitrary point within a 3D domain is located within a 3D object allows sampling the volume of the object. The fast Poisson disk sampling algorithm in [B07] can again serve as a foundation for generating samples that maintain a minimum distance and exhibit blue noise characteristics in the frequency domain. As previously mentioned, these attributes are crucial for the initial positions of particles in simulations.

Algorithm 2: Volume sampling of arbitrary 3D objects

Data: V_{obj}, l_{dst}, N_t
Result: $\mathbb{S}_{\text{obj}}$

initialize empty sample grid `s_grid` and empty active list `a_list`
define initial sample $\boldsymbol{x}_0$ inside V_{obj}
place $\boldsymbol{x}_0$ in `s_grid` and add to $\mathbb{S}_{\text{obj}}$ and `a_list`
while `a_list` not empty **do**
    `found` $\leftarrow$ **false**
    pick random $\boldsymbol{x}_i$ from `a_list`
    for $k \leftarrow 0$ **to** N_t **do**
        $r \leftarrow l_{\text{dst}}(\sqrt[3]{\rho_{\text{rnd1}}} + 1)$
        $\theta \leftarrow \arccos(2\rho_{\text{rnd2}} - 1)$
        $\phi \leftarrow 2\pi\rho_{\text{rnd3}}$
        $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_i +$ spherical2cartesian(r, θ, ϕ)
        `counter` $\leftarrow 0$
        if $\boldsymbol{x}_k$ **not** inside V_{obj} **then**
            continue to next iteration of k
        `/* check if` $\boldsymbol{x}_k$ `has no distance conflicts in` $\mathbb{S}_{\text{obj}}$ `*/`
        forall $\boldsymbol{x}_j$ in cell of $\boldsymbol{x}_k$ and in neighboring cells to $\boldsymbol{x}_k$ in `s_grid` **do**
            if dist$(\boldsymbol{x}_k, \boldsymbol{x}_j) < l_{\text{dst}}$ **then**
                continue to next iteration of k
        `/*` $\boldsymbol{x}_k$ `is a valid new sample */`
        `found` $\leftarrow$ **true**
        place $\boldsymbol{x}_k$ in `s_grid`, `a_list`, and $\mathbb{S}_{\text{obj}}$
        break
    if not `found` **then**
        remove $\boldsymbol{x}_i$ from `a_list`
return $\mathbb{S}_{\text{obj}}$

The complete algorithm is presented in Algorithm 2 in pseudocode. Similarly to Algorithm 1, a grid structure is maintained to determine neighboring sample points efficiently. Furthermore, an active list is used to track sample points that may still have potential new samples nearby. An initial sample within the 3D object can be found by defining a random point $\boldsymbol{x}_0$ uniformly within the bounding box of the 3D object and verifying its position within the volume of the object. This sample is inserted in the active list, initiating a loop that continues until no elements remain in the active list. Each iteration selects a random sample point $\boldsymbol{x}_i$ from the active list and searches for a new sample in its vicinity.



(a) Knot-mesh (b) Squirrel

(c) Armadillo (d) Bunny

Figure 3.8: Random sample points are distributed uniformly within the volume of 3D models. The same models from Figure 3.7 are used. The diameter of the spheres corresponds to the minimum distance l_{dst} between sample points.

For the surface sampling algorithm in Section 3.3.3, a sufficiently large candidate set with candidate positions on the surface is formed in advance from which neighboring candidates are then drawn, as proposed in [BWW+10]. This is necessary because defining random samples around a point $\boldsymbol{x}_i$ uniformly on the surface is challenging, as discussed in Section 3.3.3. However, a significantly larger number of candidates would be required for volume sampling since the surface is a submanifold of the volume. In contrast, it is straightforward to choose N_t random candidate positions $\boldsymbol{x}_k$ around a point $\boldsymbol{x}_i$ uniformly distributed at a distance between l_{dst} and $2l_{\text{dst}}$, as initially proposed in [B07]. The spherical coordinates

$$r = l_{\text{dst}}(\sqrt[3]{\rho_{\text{rnd1}}} + 1)\,, \tag{3.29}$$

$$\theta = \arccos(2\rho_{\text{rnd2}} - 1)\,, \tag{3.30}$$

$$\phi = 2\pi\rho_{\text{rnd3}} \tag{3.31}$$

define such a random position around the origin using three uniform random variables ρ_{rnd1}, ρ_{rnd2}, and $\rho_{\text{rnd3}} \subset [0, 1]$. Using Equations (2.4) - (2.6), this position can be converted into Cartesian coordinates and added as a translation to $\boldsymbol{x}_i$ to define a new candidate point $\boldsymbol{x}_k$ at a distance between l_{dst} and $2l_{\text{dst}}$ from $\boldsymbol{x}_i$.

For each candidate point $\boldsymbol{x}_k$, it must first be verified whether it is still positioned within the volume. The subsequent procedure for a candidate positioned within the volume is analogous to that for a candidate point $\boldsymbol{x}_{\text{cp}}$ in Algorithm 1. Figure 3.8 shows the volume sampling created using this method for four 3D models.

3.4.2 | HCP Lattice Sampling of Arbitrary Objects

Previously, the volume was randomly sampled, ensuring that sample points maintain a minimum distance from each other, although this distance may vary. However, in some cases, it is desirable for all sample points to have an equal distance from one another. In particle-based simulations, sample points designate particles, this can achieve a constant density within the volume for particles with a constant mass. Due to their inherent symmetry and simplified collision detection, particles are typically represented as spheres. While arranging points in a cubic grid for an even sampling may appear intuitive, it is not particularly efficient regarding packing density when the sample points are spherical particles.

More efficient arrangements can be observed in nature, specifically in crystals. In crystallography, the structures known as hexagonal close packing (HCP) and face-centered

cubic (FCC) exhibit the most dense packing for arranging equal-sized spheres in 3D space. Both arrangements can occupy approximately 74 % of the volume of a unit cubic box with spheres, compared to only about 52 % of this cell in a simple cubic grid arrangement. HCP arrangements have higher anisotropy, while FCC arrangements have greater symmetry and isotropy. As mentioned at the beginning of this section, symmetrical arrangements are more prone to introducing artifacts in the simulation. Therefore, an HCP arrangement is used in the following. For a more detailed examination of sphere packings and their properties, refer to [CS99].

The arrangement of the crystalline structure is referred to as a *lattice*, as is common in the literature. This wording helps to avoid confusion with the uniform grid used for neighborhood searches. In the HCP lattice, spheres are arranged in alternating layers, with each layer shifted relative to the one below. Each sphere is surrounded by six hexagonally arranged nearest neighbors within a single layer. Such a layer can be constructed by first arranging spheres in a row, e.g., along the x-axis. The distance between two spheres in this row is equal to the diameter of the spheres, denoted as l_{dst} to maintain consistency with the minimum distance between samples in previous parts. A plane, e.g., the xz-plane, can be packed more densely if successive rows in the z-direction are shifted by $l_{\text{dst}}/2$. When the rows are then packed as closely together as possible, the centers of two adjacent spheres form an equilateral triangle with the center of their mutual neighbor from a neighboring row. Therefore, according to the rule of Pythagoras, the distance between two rows is $\frac{l_{\text{dst}}}{2}\sqrt{3}$. Consequently, a position

$$\boldsymbol{x}_{\text{lc}_{xz}}(i,k) = \begin{pmatrix} (i + \frac{1}{2}(k \pmod 2)) \\ 0 \\ \frac{\sqrt{3}}{2}k \end{pmatrix} l_{\text{dst}} \tag{3.32}$$

in the 2D hexagonal lattice in the xz-plane at $y = 0$ is defined by two lattice indices i and k.

If two such layers are stacked along the remaining axis, e.g., along the y-axis, the packing density can be increased by shifting the layers relative to one another. A sphere from one layer can be placed closest to its lower layer if positioned between three spheres of the lower layer that form an equilateral triangle. The centers of the four spheres form a regular tetrahedron. The height of such a tetrahedron of $\frac{\sqrt{6}}{3}l_{\text{dst}}$ corresponds to the distance between the two layers along the y-axis. The shift in the x-axis is equivalent to $l_{\text{dst}}/2$. At the same time, the displacement in the z-axis of $\frac{\sqrt{3}}{6}l_{\text{dst}}$ corresponds to the

incircle radius of the equilateral triangle that forms the basis of the tetrahedron. This arrangement is repeated in an alternating stacking pattern. Consequently, a position

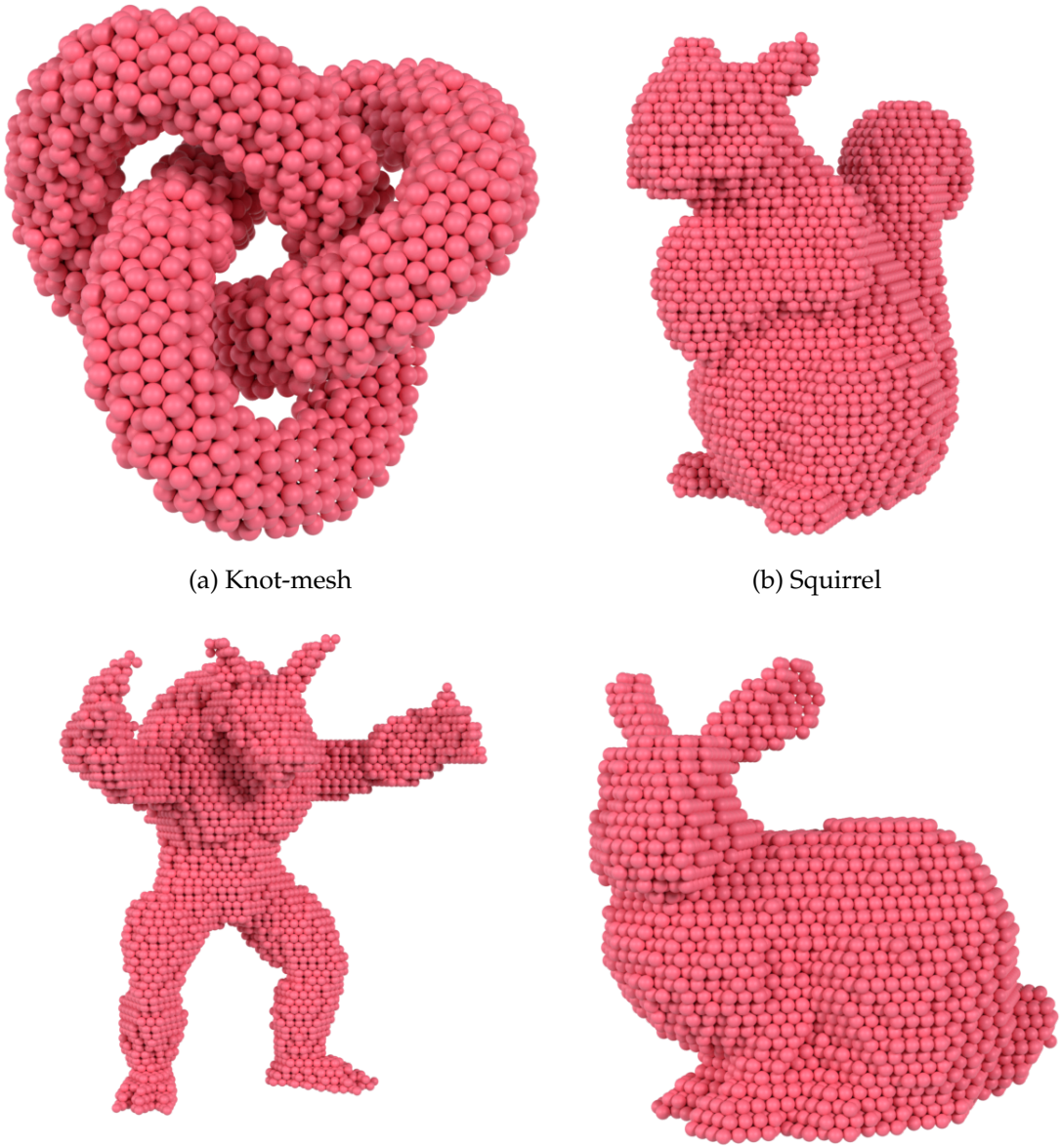
$$\boldsymbol{x}_{\mathrm{lc}}(i,j,k) = \begin{pmatrix} \left(i + \frac{1}{2}\left((j+k) \pmod 2\right)\right) \\ \frac{\sqrt{6}}{3}\, j \\ \frac{\sqrt{3}}{2}\left(k + \frac{1}{3}(j \pmod 2)\right) \end{pmatrix} l_{\mathrm{dst}} \tag{3.33}$$

in the 3D HCP lattice may be defined by three lattice indices i, j, and k.

Algorithm 3: HCP Volume sampling of arbitrary 3D objects

Data: V_{obj}, l_{dst}
Result: $\mathbb{S}_{\mathrm{obj}}$
`/* calculate lattice size in each axis from bounding box */`
`lattice_x` $\leftarrow \lceil b_{o_x} / l_{\mathrm{dst}} \rceil$
`lattice_y` $\leftarrow \lceil \frac{3}{\sqrt{6}} b_{o_y} / l_{\mathrm{dst}} \rceil$
`lattice_z` $\leftarrow \lceil \frac{2}{\sqrt{3}} b_{o_z} / l_{\mathrm{dst}} \rceil$
forall lattice cells with indices (i,j,k) **do**
  $x_{\mathrm{lc}} \leftarrow \left(i + \frac{1}{2}((j+k) \pmod 2)\right) l_{\mathrm{dst}}$
  $y_{\mathrm{lc}} \leftarrow \frac{\sqrt{6}}{3}\, j\, l_{\mathrm{dst}}$
  $z_{\mathrm{lc}} \leftarrow \frac{\sqrt{3}}{2}\left(k + \frac{1}{3}(j \pmod 2)\right) l_{\mathrm{dst}}$
  $\boldsymbol{x}_{\mathrm{lc}} \leftarrow (x_{\mathrm{lc}}, y_{\mathrm{lc}}, z_{\mathrm{lc}})^{\mathsf{T}}$
  if $\boldsymbol{x}_{\mathrm{lc}}$ **not** inside V_{obj} **then**
    continue to next cell
  place $\boldsymbol{x}_{\mathrm{lc}}$ in $\mathbb{S}_{\mathrm{obj}}$
return $\mathbb{S}_{\mathrm{obj}}$

Using this lattice, the volume of a 3D object can be sampled as densely as possible while maintaining a minimum distance of l_{dst} between sample points, assuming they are represented as spherical objects. The complete algorithm is presented in Algorithm 3 in pseudocode. Initially, the domain, limited by the object's bounding box, is subdivided into lattice cells. Unlike the grid subdivision used in previous sections, the lattice resolution varies along different coordinate axes, as described above. This subdivision determines the total number of lattice cells in each axis. For each lattice cell, it is then necessary to check whether the sample position within the cell is within the volume. Figure 3.9 shows the dense volume sampling created using this method for four 3D models.



(a) Knot-mesh (b) Squirrel

(c) Armadillo (d) Bunny

Figure 3.9: Sample points are arranged at an HCP lattice within the volume of 3D models. The same models and minimum distances l_{dst} as in Figure 3.8 are used. The diameter of the spheres is equal the minimum distance l_{dst} between sample points.

4

Simulation

The creation of virtual scenes becomes increasingly engaging when virtual objects begin to move. The transition from static to dynamic scenes is achieved through animation, which breathes life into the virtual scene. However, to ensure that these animations are convincing and adhere to the principles of the physical world, they must be based on accurate simulation techniques.

In this context, the term *simulation* describes a process whereby the behavior of real-world objects or phenomena is imitated over time. A form of simulation has already been discussed in Section 2.3, where the physically accurate description of the flow of light is presented. There, various models of differing complexity are listed, depending on the phenomena to be simulated. Similarly, the simulation of object movement can be based on models of varying complexity. High-fidelity simulations aim to replicate physical behaviors precisely, but they are computationally intensive. Such high realism is necessary, for example, in engineering applications that depend on accurate force calculations. However, for interactive applications in MR, less accurate simulations that mimic plausible behavior but can be computed in real time are sufficient.

The challenges of balancing realism and computational efficiency are particularly evident in the simulation of granular materials, such as grains, sand, and gravel. The macroscopic behavior of these materials results from the complex interactions between countless microscopic individual grains. For example, a cubic centimeter of sand consists of up to five million individual sand grains [D16], making it computationally infeasible to simulate each grain and its interactions with its neighbors at interactive frame rates. For real-time applications, it is, therefore, necessary to use a simplified approach to model the essential characteristics of granular flow at a convincing level of realism while maintaining moderate computational complexity to ensure interactive calculation times.

4.1 | Outline

This chapter introduces a novel particle-based simulation method for granular materials capable of achieving interactive frame rates. The simulation process is divided into two distinct, decoupled stages. The first stage involves accurately simulating a relatively small number of particles, including the effects of collisions and friction. In the second stage, a significantly larger number of particles is simulated based solely on the results of the first stage, neglecting collisions between these new particles. This method was initially presented in [SSS22].

Section 4.2 provides a fundamental overview of the simulation of granular material and highlights key publications in the field. Section 4.3 outlines how a small number of particles are accurately simulated in the first stage of the simulation. Section 4.4 shows how these particles are used as the foundation for simulating the larger number of particles used for visualization. Section 4.5 explains how the two stages are combined and provides insight into the implementation of the entire simulation. Finally, Section 4.6 presents simulation results visually and provides a performance evaluation that compares the method to other approaches.

4.2 | Related Work and Background

Materials such as gases, liquids, and granular substances can be described through flow fields. In physics, these flow fields are modeled within the domain of fluid dynamics. When simulating such flows, two different perspectives on the system exist.

In the *Eulerian* perspective, named after the mathematician Leonhard Euler, a fixed volume in space is observed to analyze how material properties, such as density, pressure, temperature, change as the material moves through the volume. The material properties are monitored at a finite number of fixed observation points, and values at any point in the space can be interpolated from these observation points. This perspective is well-suited for analyzing flow fields involving large-scale movements. A well-known example using the Eulerian perspective is weather modeling, in which air mass movements are simulated based on data from fixed weather stations. Notable Eulerian approaches include the finite volume method [L02] and the lattice Boltzmann method [MZ88]. For further information on Eulerian methods, refer to [KKK+17]. However, for smaller-scale systems where the movement of individual elements is more significant, a different perspective—the Lagrangian—is more appropriate.

Diameter (mm)	Size class	Category
256 <	Bowlder	Gravel
64 - 256	Cobble	
4 - 64	Pebble	
2 - 4	Granule	
1 - 2	Very coarse sand grain	Sand
1/2 - 1	Coarse sand grain	
1/4 - 1/2	Medium sand grain	
1/8 - 1/4	Fine sand grain	
1/16 - 1/8	Very fine sand grain	
1/256 - 1/16	Silt particle	Silt
< 1/256	Clay particle	Mud

Table 4.1: Wentworth sediment grain size classification as defined in [W22].

In the *Lagrangian* perspective, named after the mathematician Joseph-Louis Lagrange, the trajectories of individual elements are observed over time. The individual elements in a granular material are grains. In geology, these individual grains are typically considered rounded fragments, classified by their diameter using the Wentworth scale [W22] (see Table 4.1). Individual grains can be conceptualized as ideal spherical objects, referred to as *particles*. Each particle is characterized by a specific position, radius, and mass. Depending on whether these particles are used to infer properties about the flow field or stand-alone, a further distinction is made between continuum methods and discrete methods.

In Lagrangian continuum methods, field quantities (e.g., the velocity field) are interpolated from the corresponding quantities (e.g., the velocity of individual particles). Here, particles serve as tools to propagate attributes through the continuum. The particle size does not necessarily correspond to the size of individual elements. In particular, materials with small grain sizes, such as silt or clay (see Table 4.1), would otherwise result in impractically large numbers of particles. This practice becomes even more important in Lagrangian simulations of fluids or gases, where individual molecules are even smaller. However, decoupling the simulation resolution (i.e., the particle size) from the grain size

results in the loss of finer details at surfaces and free-falling grains. Although continuum methods are more commonly used in the simulation of liquids, a number of fluid simulation methods have been adapted to granular materials. The fluid implicit particle (FLIP) method [BKR88] can simulate sand when modeled as a continuous fluid with inner friction, cohesion, and boundary friction [ZB05]. Liquids and granular materials can be jointly simulated by modeling granular materials as porous flow [LD09] using the smoothed particle hydrodynamics (SPH) method [GM77], initially developed in astrophysics. There are also independent continuum formulations for granular materials, such as in [NGL10], where internal pressures and frictional stress are modeled with a unilateral incompressibility constraint. This formulation was subsequently integrated into predictive-corrective incompressible SPH (PCI-SPH) [SP09; AO11], an enhanced version of SPH for fluids. The material point method (MPM) [SCS94], which can be considered an enhanced version of the FLIP method, is a robust simulation method for simulating the interactions between gases, fluids, and solids [JST+16]. By applying the Drucker-Prager yield criterion [DP52], sand can be simulated within MPM [KGP+16]. With moving least squares MPM (MLS-MPM) [HFG+18], the interaction with dynamic rigid bodies can be simulated inside this continuum method.

In contrast, discrete methods describe the macroscopic behavior of the material not based on field quantities but on the microscopic interactions between individual particles. Particularly for granular simulations with fewer particles, the observed material behavior is less continuous because the grain size is coarser or because the simulation is focused on a small domain. To accurately describe such behavior, it is essential to incorporate the collisions and resulting frictional forces between individual particles. The most accurate methods that account for all forces between individual particles are based on the discrete element method (DEM) [CS79]. Due to their computational complexity, these methods are not real-time capable and are, therefore, primarily used in engineering applications. Subsequently, simplified methods derived from DEM principles were developed and used in computer graphics [BYM05], although they remained far from achieving interactive frame rates. The breakthrough towards interactive applications marked position-based dynamics (PBD) [MHH+07], simplifying calculations by resolving constraints on particle positions rather than relying on accelerations due to physical forces. Run times of PBD have been reduced by parallelizing the constraint projection scheme [MMC+14]. Further constraints have been developed for PBD that enable plausible simulation of granular materials through contact friction and cohesion [H14]. Subsequently, Coulomb friction models were introduced to simulate dynamic rigid and flexible bodies within PBD [FM17]. PBD is a widely used framework for interactive

particle-based simulation and is subject to ongoing development [MMC16; MMC+20; SC23].

Collision detection is a significant bottleneck in discrete methods for simulating granular materials. Large particle numbers are necessary to represent granular behavior accurately even when simulating coarse-grained materials. Furthermore, as the size of the particles decreases, the simulation time step must also be reduced to capture collisions correctly. Coupling particle size with time step in discrete methods is a significant challenge to computational resources, making it difficult to achieve interactive performance for fine-grained granular materials. This issue is particularly pronounced when limited to constrained hardware resources, as often in MR applications. Here, even the simulation of coarse-grained materials can be challenging.

The method presented in this chapter addresses these limitations by dividing the simulation into two distinct stages. In the first stage, the granular material is simulated using a discrete constraint-based approach that incorporates all collisions and friction effects between individual particles. In this case, the particles represent larger conglomerates of multiple grains, allowing for a fast, low-resolution simulation with larger particle radii. In the second stage, a refinement algorithm replaces the results of the first stage with a significantly larger number of particles with smaller radii corresponding to individual grains. At the time of writing, no other discrete interactive method is known to use such a division to simulate granular materials. However, alternative approaches involve the simulation of guide particles using continuum methods. These guide particles serve as a per-frame reference for rendering particles surrounding the guide particles [NGL10] or calculating the flow field of continuously moving smaller particles used only for visualization [ATO09]. Further, in [IWT12], an initial low-resolution SPH simulation is used, and the movement of smaller visualization particles is calculated based on the particles from this simulation, incorporating both the velocity field and external forces. Since the underlying continuum methods are not real-time capable, these approaches are unsuitable for interactive applications, in contrast to the method discussed in this chapter.

The approach introduced in this chapter was later modified in [GK24] by incorporating an adaptive adjustment of the particle number within the refinement stage while obeying energy conservation laws to keep the total energy constant.

4.3 | Low-resolution Simulation

As previously mentioned, the method for simulating granular materials presented in this chapter is divided into two decoupled stages. The first stage, detailed in this section, involves a discrete Lagrangian simulation. The individual elements (i.e., particles) of this stage do not correspond to the individual elements of the granular material (i.e., individual grains) but rather to a larger volume such as an aggregation of multiple grains. Given the lower resolution of this stage, these elements are referred to as low-resolution particles, in contrast to the individual elements of the second stage in Section 4.4, which correspond to the individual grains and are designated as high-resolution particles. It should be noted that throughout this section, any elements referred to as *particles* are always low-resolution particles.

The granular material comprises N_{lr} low-resolution particles, each of which—for simplicity—has the same radius r_{lr}. Additionally, each particle with index $i \in \{1, \dots, N_{lr}\}$ has a position $\boldsymbol{x}_i$ and a mass m_i. The initial positions of the individual particles can be found, for example, by sampling the initial volume of the granular material as described in Section 3.4.1. According to Newton's second law of motion, a particle with index i experiences an acceleration

$$\boldsymbol{a}_i = \frac{\boldsymbol{f}_{\text{tot},i}}{m_i} \tag{4.1}$$

due to the forces $\boldsymbol{f}_{\text{tot},i}$ acting on it. This relationship is the foundation for describing the particle's motion. The particle's velocity $\boldsymbol{v}_i$ evolves in time according to:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v}_i = \boldsymbol{a}_i\,, \tag{4.2}$$

while its position $\boldsymbol{x}_i$ changes as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}_i = \boldsymbol{v}_i\,. \tag{4.3}$$

To describe the particle's trajectory, these quantities must be integrated over time. Analytical solutions to these integrations are often impractical for complex systems, which is why numerical integration methods are used to approximate the trajectory. Standard methods include the Euler integration, the Verlet integration [V67], and the Runge-Kutta methods [K01]. For more detailed information on numerical techniques for integrating Newton's equations of motion, refer to [PTV+07].

The acting forces can be classified into two categories: globally applicable external forces $\boldsymbol{f}_{\text{ext}}$ and local forces acting on individual particles. Local forces act within the granular material on individual particles due to collisions, friction, adhesion, and cohesion. Global

external forces act on the entire system from outside. Here, the only global force of significance is the Earth's gravitational force $\boldsymbol{f}_g$, which results in a mass- and time-independent acceleration of $\boldsymbol{g}_e \approx -9.81\,\mathrm{m\,s^{-2}} \cdot \hat{\boldsymbol{e}}_2$. Other examples of global forces include external electric or magnetic fields acting on charged particles or wind forces affecting particles in cloth dynamics or atmospheric simulations.

Simulating all locally acting forces is computationally challenging in systems with many tiny particles. Force-based simulations, such as DEM [CS79], require very small time steps Δt_{lr} to ensure numerical stability. A common practice in real-time simulations is to exclude the acceleration and velocity layers and instead apply changes directly at the position level. In position-based simulations, such as PBD [MHH+07], each time step $t_n \rightarrow t_n + \Delta t_{lr}$ initially allows each particle to undergo unconstrained motion, resulting in a new theoretical predicted position. In the following, $\boldsymbol{x}^*$ will denote predicted positions, while $\boldsymbol{x}$ will continue to denote final positions. The predicted position

$$\boldsymbol{x}_i^*(t_n + \Delta t_{lr}) = \boldsymbol{x}_i(t_n) + \Delta t_{lr} \boldsymbol{v}_i^*(t_n + \Delta t_{lr}) \tag{4.4}$$

of a particle with index i can be calculated through a symplectic Euler integration based on its unconstrained velocity

$$\boldsymbol{v}_i^*(t_n + \Delta t_{lr}) = \boldsymbol{v}_i(t_n) + \Delta t_{lr} \frac{\boldsymbol{f}_{\text{ext}}}{m_i} \tag{4.5}$$

at time $t_n + \Delta t_{lr}$, as proposed in [MHH+07]. The predicted position does not yet account for the effects of local forces. These effects are subsequently incorporated by iteratively adjusting the predicted position through positional corrections $\Delta \boldsymbol{x}_i$, enforcing various constraints (e.g., the minimum distance between individual particles). Once all constraints have been satisfied, the resulting velocity

$$\boldsymbol{v}_i(t_n + \Delta t_{lr}) = \frac{\boldsymbol{x}_i^*(t_n + \Delta t_{lr}) + \Delta \boldsymbol{x}_i - \boldsymbol{x}_i(t_n)}{\Delta t_{lr}} \tag{4.6}$$

can be calculated for the time $t_n + \Delta t_{lr}$. In the following, the indication of time is omitted for reasons of readability, as it is self-evident from the distinction between position $\boldsymbol{x}_i$ and predicted position $\boldsymbol{x}_i^*$. Although such a position-based approach sacrifices some physical accuracy, it enables a more stable and faster simulation, as well as the use of larger time steps.

4.3.1 | Constraints

The essence of position-based simulation techniques lies in the use of constraints. Position changes induced by constraint projection replace those that would otherwise result from

local forces. A constraint can affect any number $N_{p,c} \geq 1$ of particles. Mathematically, a constraint is defined as a scalar function

$$c_t : \mathbb{R}^{3N_{p,c}} \longrightarrow \mathbb{R} \tag{4.7}$$

of the predicted positions $\boldsymbol{X} = (\boldsymbol{x}_1^{*\mathsf{T}}, \ldots, \boldsymbol{x}_{N_{p,c}}^{*\mathsf{T}})$ of all involved particles, mapping the set of predicted particle positions to a real number. The function value must meet a specific condition for a constraint to be satisfied. PBD distinguishes two types of constraints: equality constraints $c_t(\boldsymbol{X}) = 0$ and inequality constraints $c_t(\boldsymbol{X}) \geq 0$.

If a constraint is violated, the goal of *constraint projection* is to find small positional corrections $\Delta\boldsymbol{X} = (\Delta\boldsymbol{x}_1^{\mathsf{T}}, \ldots, \Delta\boldsymbol{x}_{N_{p,c}}^{\mathsf{T}})$ applied to all particles involved in the constraint that ensure the condition is once again fulfilled. Since these positional corrections are small, a first-order Taylor expansion

$$c_t(\boldsymbol{X} + \Delta\boldsymbol{X}) \approx c_t(\boldsymbol{X}) + \nabla_{\boldsymbol{X}} c_t(\boldsymbol{X})^{\mathsf{T}} \Delta\boldsymbol{X} + \mathcal{O}(\Delta\boldsymbol{X}^2) \tag{4.8}$$

around $\boldsymbol{X}$ can be used as a linear approximation of $c_t(\boldsymbol{X} + \Delta\boldsymbol{X})$. Setting this approximation (Equation (4.8)) to 0 leads to an under-determined optimization problem for finding the positional corrections $\Delta\boldsymbol{X}$. In [MHH+07], it is demonstrated that by imposing the condition that the direction of the positional corrections is aligned with the constraint gradient, both linear and angular momentum are conserved. A violation of momentum conservation would introduce ghost forces into the system, leading to physically incorrect results. By applying this condition, the positional corrections

$$\Delta\boldsymbol{X} = \lambda_X \nabla_{\boldsymbol{X}} c_t(\boldsymbol{X}) \tag{4.9}$$

can be formulated using a Lagrange multiplier λ_X. Substituting (4.9) into (4.8), solving for λ_X, and then substituting λ_X into (4.9), yields:

$$\Delta\boldsymbol{X} = -\frac{c_t(\boldsymbol{X})}{|\nabla_{\boldsymbol{X}} c_t(\boldsymbol{X})|^2} \nabla_{\boldsymbol{X}} c_t(\boldsymbol{X}) \tag{4.10}$$

for the positional corrections. This yields the positional correction

$$\Delta\boldsymbol{x}_i = -\frac{c_t(\boldsymbol{X})}{|\nabla_{\boldsymbol{X}} c_t(\boldsymbol{X})|^2} \nabla_{\boldsymbol{x}_i^*} c_t(\boldsymbol{X}) = -\frac{c_t(\boldsymbol{X})}{\sum_k^{N_{p,c}} \left| \nabla_{\boldsymbol{x}_k^*} c_t(\boldsymbol{X}) \right|^2} \nabla_{\boldsymbol{x}_i^*} c_t(\boldsymbol{X}) \tag{4.11}$$

for a particle with index i involved in the constraint.

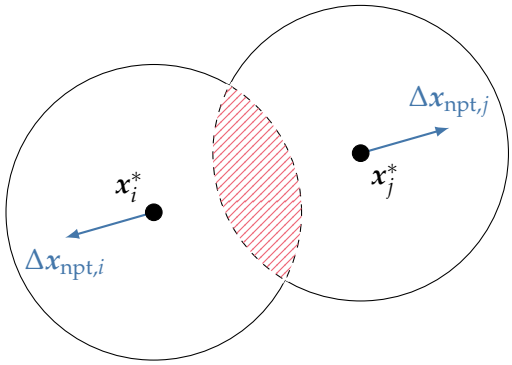


Figure 4.1: A collision between particles with indices i and j at predicted positions x_i^* and x_j^*. The non-penetration constraint applies positional corrections $\Delta x_{\text{npt},i}$ and $\Delta x_{\text{npt},j}$ to resolve the interpenetration (illustrated in red), thereby simulating the collision behavior.

Here, the masses of the individual particles have not yet been considered. If all particles have the same mass, this approach is valid. However, the corrections will violate momentum conservation if particles have different masses. Momentum conservation is maintained if the positional correction Δx_i is scaled by the inverse of the particle mass m_i, such that:

$$\Delta x_i = -\frac{1}{m_i} \frac{c_t(\boldsymbol{X})}{\sum_k^{N_{p,c}} \frac{1}{m_k} \left| \nabla_{x_k^*} c_t(\boldsymbol{X}) \right|^2} \nabla_{x_i^*} c_t(\boldsymbol{X}) \,. \tag{4.12}$$

This is important, as the particles in the granular simulation will be assigned different masses for computational purposes during contact handling, as explained later.

The behavior of granular material is primarily characterized by the contact interactions between individual elements, with collisions between particles playing a crucial role. Within a time step, a collision between two particles with indices i and j occurs when the (Euclidean) distance

$$|x_{ij}^*| = |x_j^* - x_i^*| = \sqrt{(x_j^* - x_i^*)^\mathsf{T}(x_j^* - x_i^*)} \tag{4.13}$$

between their predicted positions x_i^*, x_j^* is less than the sum of their radii r_{lr} (see Figure 4.1). The non-penetration constraint

$$c_{\text{npt}}(x_i^*, x_j^*) = |x_{ij}^*| - 2r_{lr} \geq 0 \tag{4.14}$$

prevents interpenetration between the two colliding particles. Using the gradients $\nabla_{x_i^*} c_{\text{npt}} = -\hat{n}_{ij}$ and $\nabla_{x_j^*} c_{\text{npt}} = \hat{n}_{ij}$ (see Appendix A.2 for a detailed derivation) with $\hat{n}_{ij} =$

$\boldsymbol{x}_{ij}^*/|\boldsymbol{x}_{ij}^*|$ being the collision normal, positional corrections

$$\Delta \boldsymbol{x}_{\text{npt},i} = \frac{\frac{1}{m_i}}{\frac{1}{m_i}+\frac{1}{m_j}} \left(|\boldsymbol{x}_{ij}^*| - 2r_{lr}\right) \hat{\boldsymbol{n}}_{ij}\,, \tag{4.15}$$

$$\Delta \boldsymbol{x}_{\text{npt},j} = -\frac{\frac{1}{m_j}}{\frac{1}{m_i}+\frac{1}{m_j}} \left(|\boldsymbol{x}_{ij}^*| - 2r_{lr}\right) \hat{\boldsymbol{n}}_{ij} \tag{4.16}$$

for the two colliding particles can be derived according to Equation (4.12).

The non-penetration constraint models collisions between individual particles, ensuring the conservation of momentum and energy. With this constraint alone, a collision between two particles behaves similarly to a collision between two billiard balls. However, real grains, with their surface irregularities, interact differently, as energy is reduced through the effects of friction during each contact. Friction allows the granular material to form stable piles and prevents the particles from moving away from each other indefinitely. A simple model for describing friction is Coulomb's law of friction [P10], characterizing friction as a force acting tangentially to the contact normal, opposing the motion.

Coulomb's law in a position-based formulation is equivalent to a friction constraint reducing positional changes tangential to the contact normal $\hat{\boldsymbol{n}}_{ij}$. Consequently, friction should be applied following the contact between two particles with indices i and j once the positional changes $\Delta \boldsymbol{x}_{\text{npt},i}$ and $\Delta \boldsymbol{x}_{\text{npt},j}$ due to the collision have already been applied to the predicted positions. This results in a relative displacement

$$\boldsymbol{d}_{ij} = (\boldsymbol{x}_j^* - \boldsymbol{x}_j) - (\boldsymbol{x}_i^* - \boldsymbol{x}_i) \tag{4.17}$$

between the current predicted positions $\boldsymbol{x}_i^*$ and $\boldsymbol{x}_j^*$ and the initial positions $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ at the beginning of the time step. This displacement can be split into a component $(\boldsymbol{d}_{ij}^\mathsf{T}\hat{\boldsymbol{n}}_{ij})\hat{\boldsymbol{n}}_{ij}$ along the contact normal and the projection

$$\boldsymbol{d}_{ij_\perp} = (\boldsymbol{d}_{ij}^\mathsf{T}\hat{\boldsymbol{t}}_{ij})\hat{\boldsymbol{t}}_{ij} = \boldsymbol{d}_{ij} - (\boldsymbol{d}_{ij}^\mathsf{T}\hat{\boldsymbol{n}}_{ij})\hat{\boldsymbol{n}}_{ij} \tag{4.18}$$

onto the plane tangent to the contact normal, with a unit vector $\hat{\boldsymbol{t}}_{ij}$ tangential to $\hat{\boldsymbol{n}}_{ij}$. Therefore, the total motion tangential to the contact normal can be restricted by the constraint

$$c_f(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*) = \boldsymbol{d}_{ij}^\mathsf{T}\hat{\boldsymbol{t}}_{ij} = 0\,. \tag{4.19}$$

The directions of the contact normal $\hat{\boldsymbol{n}}_{ij}$ and the tangent $\hat{\boldsymbol{t}}_{ij}$ remain constant under positional changes in the tangential direction, which leads to the gradients $\nabla_{\boldsymbol{x}_i^*} c_f = -\hat{\boldsymbol{t}}_{ij}$ and

$\nabla_{\boldsymbol{x}_j^*} c_f = \hat{\boldsymbol{t}}_{ij}$. Following Equation (4.12), the positional corrections

$$\Delta \boldsymbol{x}_{t,i} = \frac{\frac{1}{m_i}}{\frac{1}{m_i} + \frac{1}{m_j}} (\boldsymbol{d}_{ij}^{\mathsf{T}} \hat{\boldsymbol{t}}_{ij}) \hat{\boldsymbol{t}}_{ij} = \frac{\frac{1}{m_i}}{\frac{1}{m_i} + \frac{1}{m_j}} \boldsymbol{d}_{ij_\perp}, \tag{4.20}$$

$$\Delta \boldsymbol{x}_{t,j} = -\frac{\frac{1}{m_j}}{\frac{1}{m_i} + \frac{1}{m_j}} (\boldsymbol{d}_{ij}^{\mathsf{T}} \hat{\boldsymbol{t}}_{ij}) \hat{\boldsymbol{t}}_{ij} = -\frac{\frac{1}{m_j}}{\frac{1}{m_i} + \frac{1}{m_j}} \boldsymbol{d}_{ij_\perp} \tag{4.21}$$

are derived to cancel out the motion tangential to the contact normal $\hat{\boldsymbol{n}}_{ij}$.

Positional corrections $\Delta \boldsymbol{x}_{t,i}$ and $\Delta \boldsymbol{x}_{t,j}$ prevent any tangential movement, corresponding to the effect of static friction. This occurs when the static friction force $\boldsymbol{f}_s$ is greater than the sum of the tangential forces $\boldsymbol{f}_t$ acting on the particle, resulting in no movement tangential to the contact normal. In the Coulomb model, the magnitude of this static friction force $|\boldsymbol{f}_s|$ is proportional to the magnitude of the force acting along the contact normal $|\boldsymbol{f}_n|$, with proportionality constant μ_s known as the *coefficient of static friction*. Therefore, static friction occurs when

$$|\boldsymbol{f}_t| \leq \mu_s |\boldsymbol{f}_n| . \tag{4.22}$$

This concept can be transferred to the position-based context, as demonstrated below for the particle with index i. The displacement $-\Delta \boldsymbol{x}_{t,i}$ represents the tangential position change within a time step Δt_{lr} before friction is applied. This corresponds to a tangential force $\boldsymbol{f}_{t,i} = -m_i \Delta \boldsymbol{x}_{t,i} / \Delta t_{lr}^2$. Additionally, if $\Delta \boldsymbol{x}_{\text{npt},i}$ is the positional correction along the contact normal that prevents interpenetration, then the contact normal force $\boldsymbol{f}_{n,i} = -m_i \Delta \boldsymbol{x}_{\text{npt},i} / \Delta t_{lr}^2$ acts in the opposing direction. This implies that if

$$|\Delta \boldsymbol{x}_{t,i}| \leq \mu_s \left| \Delta \boldsymbol{x}_{\text{npt},i} \right| , \tag{4.23}$$

static friction should act on the particle with index i, preventing tangential movement.

Kinetic friction, in contrast, results in a reduction in tangential velocity. This effect is caused by a sliding friction force that acts in opposition to the tangential motion. In the Coulomb model, the magnitude of the sliding friction force is also proportional to the magnitude of the force acting along the contact normal. In the case of kinetic friction, the proportionality constant μ_k is referred to as the *coefficient of kinetic friction*. For the majority of materials, the relationship $0 < \mu_k < \mu_s$ holds. Therefore, this assumption will be applied in the following.

Coulomb's kinetic friction model can also be applied to the position-based context. The sliding friction force for a particle i reduces tangential motion but does not entirely

(a) $\mu_s = 0.3$ (b) $\mu_s = 0.55$ (c) $\mu_s = 0.8$

Figure 4.2: Granular materials with different coefficients of friction form piles with different angles of repose.

prevent it, as was the case in Equation (4.20). Accordingly, the positional correction for sliding friction

$$\Delta \boldsymbol{x}_{k,i} = \alpha_k \Delta \boldsymbol{x}_{t,i} = \frac{\frac{1}{m_i}}{\frac{1}{m_i} + \frac{1}{m_j}} \alpha_k \boldsymbol{d}_{ij_\perp} \tag{4.24}$$

can be described by applying a scalar factor $\alpha_k \leq 1$ to the correction $\Delta \boldsymbol{x}_{t,i}$ that restricts any tangential movement. α_k can be determined by considering the edge case where the force balance $|\boldsymbol{f}_t| = |\boldsymbol{f}_k|$ is even and tangential motion is stopped. As before, the force balance

$$m_i \frac{|\Delta \boldsymbol{x}_{t,i}|}{\Delta t_{lr}^2} = m_i \frac{\mu_k |\Delta \boldsymbol{x}_{\text{npt},i}|}{\Delta t_{lr}^2} \tag{4.25}$$

can be expressed by the resulting forces from the positional corrections. In the edge case, further $\alpha_k = 1$ holds and from Equation (4.25) follows $\mu_k |\Delta \boldsymbol{x}_{\text{npt},i}| / |\Delta \boldsymbol{x}_{t,i}| = 1$, leading to $\alpha_k = \mu_k |\Delta \boldsymbol{x}_{\text{npt},i}| / |\Delta \boldsymbol{x}_{t,i}|$.

Static and kinetic friction can be unified into single positional corrections for general friction

$$\Delta \boldsymbol{x}_{f,i} = \frac{\frac{1}{m_i}}{\frac{1}{m_i} + \frac{1}{m_j}} \boldsymbol{d}_{ij_\perp} \cdot \begin{cases} 1 & \text{if} \quad |\Delta \boldsymbol{x}_{t,i}| \leq \mu_s |\Delta \boldsymbol{x}_{\text{npt},i}| \\ \min\left(\frac{\mu_k |\Delta \boldsymbol{x}_{\text{npt},i}|}{|\Delta \boldsymbol{x}_{t,i}|}, 1 \right) & \text{else} \end{cases}, \tag{4.26}$$

$$\Delta \boldsymbol{x}_{f,j} = -\frac{\frac{1}{m_j}}{\frac{1}{m_i} + \frac{1}{m_j}} \boldsymbol{d}_{ij_\perp} \cdot \begin{cases} 1 & \text{if} \quad |\Delta \boldsymbol{x}_{t,j}| \leq \mu_s |\Delta \boldsymbol{x}_{\text{npt},j}| \\ \min\left(\frac{\mu_k |\Delta \boldsymbol{x}_{\text{npt},j}|}{|\Delta \boldsymbol{x}_{t,j}|}, 1 \right) & \text{else} \end{cases} \tag{4.27}$$

for the particles with indices i and j. If the resulting friction force is greater than the resulting tangential force, the minimum function ensures that no negative positional change occurs, enforcing static friction instead. This friction model is closely related to those proposed in [MMC+14] and [H14].

The coefficient of static friction μ_s is geometrically related to the minimum angle γ_f of an inclined plane at which a body would begin to slide down by $\mu_s = \tan \gamma_f$ [M08].

Consequently, the maximum angle of a pile formed of granular material—the *angle of repose*—is correlated with μ_s. The rougher the material, the higher the coefficient of friction and, consequently, the larger the angle of repose. Typical angles of repose for granular materials range from 15° for wet silt to 45° for coarse gravel [BB18]. Accordingly, the coefficient of static friction μ_s can range from 0.25 for wet silt to 1.0 for coarse gravel. In Figure 4.2, granular materials with different values of μ_s are simulated using the positional corrections from Equations (4.26) and (4.27), resulting in piles with varying angles of repose. In reality, γ_f and the angle of repose differ, as factors like adhesion also influence the angle of repose. In [H14], it is demonstrated how adhesion can also be formulated as a constraint. However, in the context of this thesis, the effects of adhesion are considered negligible.

It is insufficient to simulate only the interactions between the individual particles of the granular material. Without interactions with domain boundaries or other objects, the particles would fall down, accelerating indefinitely due to gravity. A straightforward approach involves representing the surfaces of domain boundaries and static rigid bodies with particles [AIA+12]. Surface sampling, as described in Section 3.3.3, can be used for this purpose. By assigning an infinitely large mass to these static particles, the same distance and friction positional corrections as in Equations (4.15) and (4.26) can be applied to model collisions between a granular particle with index i and a static particle with index j. Consequently, the positional corrections for the static particles are zero, since $\frac{1}{m_j} \xrightarrow[m_j \to \infty]{} 0$. As the static particles influence the simulated particles, but not vice versa, this interaction is referred to as *one-way coupling*.

Collisions with static objects already allow for the simulation of many interesting scenes, but things become truly compelling when interactions with other dynamic objects are also modeled. When contact forces act on both the simulated particles and the dynamic object, this is referred to as *two-way coupling*. Dynamic objects, or dynamic rigid bodies, to be more specific, can also be modeled within the position-based context. Therefore, the volume of the rigid body is represented by particles with the same radius r_{lr} as the rest of the simulation. A volume sampling method, as described in Section 3.4.2, can be used for this purpose. In the following, each rigid body is assumed to have a homogeneous density (i.e., each particle represents an equal portion of the object's total mass).

A rigid body comprises N_{rbo} particles in an initial undeformed configuration at positions $\boldsymbol{x}_i(t_0) := \boldsymbol{x}_i^{(0)}$ $(i \in \{1, \ldots, N_{rbo}\})$ at the start of the simulation. During the simulation, these particles move to new estimated positions $\boldsymbol{x}_i^*$ due to external forces, velocities, and

positional corrections. This movement results in a new, currently deformed configuration. As the rigid body is not meant to deform, positional corrections $\Delta \boldsymbol{x}_{\text{rbo},i}$ must be found, so the corrected configuration given by positions $\boldsymbol{x}_i^* + \Delta \boldsymbol{x}_{\text{rbo},i}$ corresponds to the result of a rigid motion—specifically, a rotation and translation—applied to the initial undeformed configuration. The first step is to use shape matching [MHT+05] to identify the rigid motion that best aligns the initial and the current configurations. By setting both systems relative to their centroids

$$\boldsymbol{c}_{\text{rbo}}^{(0)} = \frac{\sum_i \boldsymbol{x}_i^{(0)}}{N_{\text{rbo}}}, \tag{4.28}$$

$$\boldsymbol{c}_{\text{rbo}} = \frac{\sum_i \boldsymbol{x}_i^*}{N_{\text{rbo}}} \tag{4.29}$$

only the rotation $\boldsymbol{R}_{\text{rbo}}$ needs to be determined, which can be formulated as an optimization problem:

$$\min \sum_i \left| \boldsymbol{R}_{\text{rbo}}(\boldsymbol{x}_i^{(0)} - \boldsymbol{c}_{\text{rbo}}^{(0)}) - (\boldsymbol{x}_i^* - \boldsymbol{c}_{\text{rbo}}) \right|^2. \tag{4.30}$$

The two relative vectors, $\bar{\boldsymbol{x}}_i := \boldsymbol{x}_i^{(0)} - \boldsymbol{c}_{\text{rbo}}^{(0)}$ and $\bar{\boldsymbol{x}}_i^* := \boldsymbol{x}_i^* - \boldsymbol{c}_{\text{rbo}}$, will be used below to enhance readability. The unconstrained solution

$$\boldsymbol{T}_{\text{rbo}} = \left(\sum_i \bar{\boldsymbol{x}}_i^* \bar{\boldsymbol{x}}_i^\mathsf{T} \right) \left(\sum_i \bar{\boldsymbol{x}}_i \bar{\boldsymbol{x}}_i^\mathsf{T} \right)^{-1} \tag{4.31}$$

to the optimization problem (see Appendix A.3 for a detailed derivation) does not necessarily yield a rotation matrix $\boldsymbol{R}_{\text{rbo}}$ but rather a linear transformation $\boldsymbol{T}_{\text{rbo}}$. This transformation matrix may include not only rotation but also scaling and shearing. Therefore, extracting the rotational component from this linear transformation is necessary. The matrix $\sum_i \bar{\boldsymbol{x}}_i \bar{\boldsymbol{x}}_i^\mathsf{T}$ as a product of symmetric matrices $\bar{\boldsymbol{x}}_i \bar{\boldsymbol{x}}_i^\mathsf{T}$ is symmetric. The inverse of a symmetric matrix (if it is invertible) is also symmetric, and a symmetric matrix can only represent scaling and shearing. This implies that the desired rotational component of matrix $\boldsymbol{T}_{\text{rbo}}$ must be contained within matrix $\sum_i \bar{\boldsymbol{x}}_i^* \bar{\boldsymbol{x}}_i^\mathsf{T}$. The polar decomposition

$$\sum_i \bar{\boldsymbol{x}}_i^* \bar{\boldsymbol{x}}_i^\mathsf{T} = \boldsymbol{R}_{\text{rbo}} \boldsymbol{S}_{\text{rbo}} \tag{4.32}$$

allows for the decomposition of matrix $\sum_i \bar{\boldsymbol{x}}_i^* \bar{\boldsymbol{x}}_i^\mathsf{T}$ into an orthogonal matrix $\boldsymbol{R}_{\text{rbo}}$ and another symmetric matrix $\boldsymbol{S}_{\text{rbo}}$. A rotation matrix has the fundamental property of always being an orthogonal matrix, so this orthogonal matrix $\boldsymbol{R}_{\text{rbo}}$ corresponds to the desired rotation matrix. A typical approach to determine the two matrices $\boldsymbol{R}_{\text{rbo}}$ and $\boldsymbol{S}_{\text{rbo}}$ of the polar decomposition is to compute the singular value decomposition of the matrix $\sum_i \bar{\boldsymbol{x}}_i^* \bar{\boldsymbol{x}}_i^\mathsf{T}$. For a more detailed explanation and alternative methods, refer to [H86].

Figure 4.3: A rigid body (squirrel), represented by a collection of particles, is thrown into a tower of granular particles. The interaction between the granular particles and the rigid body particles is simulated, with the rigid body particles being constrained to maintain the shape of the squirrel through a shape-matching constraint. This results in a two-way coupling between the granular material and the rigid body.

Accordingly, an optimal current undeformed target configuration exists resulting from a rigid motion—comprising a rotation $\boldsymbol{R}_{\text{rbo}}$ and a translation $\boldsymbol{t}_{\text{rbo}} = \boldsymbol{c}_{\text{rbo}} - \boldsymbol{R}_{\text{rbo}}\boldsymbol{c}_{\text{rbo}}^{(0)}$—applied to the initial undeformed configuration. For each particle with index i in the rigid body, a current target position

$$\boldsymbol{q}_{\text{rbo},i} = \boldsymbol{R}_{\text{rbo}}\boldsymbol{x}_i^{(0)} + \boldsymbol{t}_{\text{rbo}} = \boldsymbol{R}_{\text{rbo}}\bar{\boldsymbol{x}}_i + \boldsymbol{c}_{\text{rbo}} \tag{4.33}$$

can be determined. Therefore, a positional correction $\Delta\boldsymbol{x}_{\text{rbo},i}$ must be identified to move each particle to its target position $\boldsymbol{q}_{\text{rbo},i}$. A positional constraint

$$c_{\text{rbo}}(\boldsymbol{x}_i^*) = |\boldsymbol{q}_{\text{rbo},i} - \boldsymbol{x}_i^*| = 0 \tag{4.34}$$

for each particle can enforce the target positions. With the gradient

$$\underset{\boldsymbol{x}_i^*}{\nabla} c_{\text{rbo}}(\boldsymbol{x}_i^*) = -\frac{\boldsymbol{q}_{\text{rbo},i} - \boldsymbol{x}_i^*}{|\boldsymbol{q}_{\text{rbo},i} - \boldsymbol{x}_i^*|} \tag{4.35}$$

(calculated analogously to $\underset{\boldsymbol{x}_i^*}{\nabla} c_{\text{npt}}$ in Appendix A.2, since the target positions $\boldsymbol{q}_{\text{rbo},i}$ are held constant), the positional correction

$$\Delta\boldsymbol{x}_{\text{rbo},i} = \boldsymbol{R}_{\text{rbo}}\bar{\boldsymbol{x}}_i + \boldsymbol{c}_{\text{rbo}} - \boldsymbol{x}_i^* \tag{4.36}$$

can be derived according to Equation (4.12).

Figure 4.3 shows frames of a scene where a rigid body is simulated alongside granular material with two-way coupling. It is important to note that such shape matching remains stable only for small deviations from the undeformed configuration, which is not an issue given the small time steps Δt_{lr} needed for the contact constraints. Region-based shape matching [MHT+05; RJ07] can be applied to larger deviations in the current deformed configuration. Additionally, there are methods where, instead of shape matching followed by per-particle constraints, a single constraint is solved for the entire rigid body, which, while accounting for the inertia tensor, is more physically accurate but computationally more complex [DCB16; MMC+20].

4.3.2 | Solver

Previously, in Section 4.3.1, different constraints were formulated to ensure that the simulated system conforms to the desired physical behavior. For each constraint, a positional correction can be derived for each particle involved. A particle can be affected by multiple constraints simultaneously. The constraints form a system that must be solved iteratively until convergence. A converged system is in a physically plausible state in which no constraints are violated.

In the original formulation of PBD [MHH+07], all constraint projections (i.e., the calculation and application of positional corrections to predicted positions) are executed sequentially, one after the other. This is equivalent to a Gauss-Seidel iteration, which has the advantage that subsequent constraints are always solved with the most recent predicted positions. Such a constraint projection typically results in fast and stable convergence. However, the serial nature of Gauss-Seidel iterations also has the disadvantage that the sequential updates are not straightforward to parallelize. Particularly for simulations of granular materials with large particle numbers, parallel constraint projection on massively parallel devices (e.g., modern mobile processors or GPUs) offers significant speed advantages.

In [MMC+14], an alternative parallel constraint projection scheme is presented. In this approach, all positional corrections based on the predicted positions at the beginning of the iteration are computed first without applying any corrections until all constraints have been evaluated. This allows the computation of positional corrections to be parallelized. Once all positional corrections have been calculated, they are simultaneously applied to the predicted positions, corresponding to a Gauss-Jacobi iteration. The Gauss-Jacobi

iteration, however, has the disadvantage that applying all corrections simultaneously often results in slower convergence, requiring more iterations. In addition, it can lead to oscillations, potentially destabilizing the entire simulation. This can be avoided by under-relaxation through constraint averaging [BFA02], where the total positional correction

$$\Delta \boldsymbol{x}_{\text{tot},i} = \frac{1}{N_{c,i}} \sum_{k=1}^{N_{c,i}} \Delta \boldsymbol{x}_{k,i} \tag{4.37}$$

for a particle with index i is averaged over the number of constraints $N_{c,i}$ acting on the particle. This relaxation ensures convergence but does not guarantee momentum conservation if contact partners have different numbers of constraints acting on them. However, the visual effects of this violation are negligible for the simulation of granular material.

4.3.3 | Algorithm

Algorithm 4 shows an entire low-resolution simulation step for a time step Δt_{lr} including the previously presented constraints. This simulation step is repeatedly executed within a simulation loop until the cumulative time steps match the interval between two frames at the desired frame rate. The resulting low-resolution positions and velocities subsequently serve as the foundation for the high-resolution simulation (see Section 4.4).

First, unconstrained velocities and predicted positions for all particles are determined using symplectic Euler integration, as described at the beginning of Section 4.3. Additionally, a reduced mass

$$m_i^* = m_i e^{-\boldsymbol{x}_i^{*\mathsf{T}} \hat{\boldsymbol{e}}_2} \tag{4.38}$$

is calculated based on the height of each particle. This reduced mass is used in the contact constraints between granular particles to achieve more stable and taller particle stacks with fewer required solver iterations, as demonstrated in [MMC+14].

Each particle's neighboring particles—and, therefore, collision partners—can be identified based on the predicted positions. This neighborhood search can be performed using a uniform grid structure as described in Section 3.3.3. Alternatively, more computationally- and memory-efficient methods exist—such as spatial hashing [THM+03], compact hashing [IAB+11], and bounding volume hierarchies [HSM+19]—but are not discussed further in this thesis. Since neighborhood relationships change only slightly between iterations within a single time step, it is more efficient to update the neighborhood search only once per simulation step. To ensure that particles displaced by small positional corrections

Algorithm 4: A single low-resolution simulation time step

Data: Positions $\boldsymbol{x}_i$ and velocities $\boldsymbol{v}_i$ at a time t_n
Result: Positions $\boldsymbol{x}_i$ and velocities $\boldsymbol{v}_i$ at a time $t_n + \Delta t_{lr}$

forall particles with index i **do**
  $\boldsymbol{v}_i^* \leftarrow \boldsymbol{v}_i + \Delta t_{lr} \boldsymbol{g}_e$
  $\boldsymbol{x}_i^* \leftarrow \boldsymbol{x}_i + \Delta t_{lr} \boldsymbol{v}_i^*$
  $m_i^* \leftarrow m_i e^{-\boldsymbol{x}_i^{*\mathsf{T}} \hat{\boldsymbol{e}}_2}$
find particle neighbors
for number of stabilization iterations **do**
  forall collision pairs $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ **do**
    $\Delta \boldsymbol{x}_{\text{npt},i}, \Delta \boldsymbol{x}_{\text{npt},j} \leftarrow c_{\text{npt}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  forall particles with index i **do**
    $\Delta \boldsymbol{x}_{\text{tot},i}, N_{c,i} \leftarrow$ accumulate all positional corrections
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \Delta \boldsymbol{x}_{\text{tot},i} / N_{c,i}$
    $\boldsymbol{x}_i^* \leftarrow \boldsymbol{x}_i^* + \Delta \boldsymbol{x}_{\text{tot},i} / N_{c,i}$
for number of solver iterations **do**
  forall collision pairs $\boldsymbol{x}_i^*$, $\boldsymbol{x}_j^*$ **do**
    $\Delta \boldsymbol{x}_{\text{npt},i}, \Delta \boldsymbol{x}_{\text{npt},j} \leftarrow c_{\text{npt}}(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*)$
    $\Delta \boldsymbol{x}_{f,i}, \Delta \boldsymbol{x}_{f,j} \leftarrow c_f(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*)$
  forall dynamic rigid bodies **do**
    $\boldsymbol{R}_{\text{rbo}}, \boldsymbol{t}_{\text{rbo}} \leftarrow$ apply shape matching
    forall $\boldsymbol{x}_i^*$ in rigid body **do**
      $\Delta \boldsymbol{x}_{\text{rbo},i} \leftarrow c_{\text{rbo}}(\boldsymbol{x}_i^*)$
  forall particles with index i **do**
    $\Delta \boldsymbol{x}_{\text{tot},i}, N_{c,i} \leftarrow$ accumulate all positional corrections
    $\boldsymbol{x}_i^* \leftarrow \boldsymbol{x}_i^* + \Delta \boldsymbol{x}_{\text{tot},i} / N_{c,i}$
forall particles with index i **do**
  $\boldsymbol{v}_i \leftarrow (\boldsymbol{x}_i^* - \boldsymbol{x}_i) / \Delta t_{lr}$
  $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i^*$

are still detected as potential collision partners, it is advisable to set the search radius slightly larger than $2r_{lr}$. Increasing the search radius by 10–20 % is usually sufficient, depending on the chosen time step. An even larger search radius is required to reuse the neighborhood search in the second stage of the simulation. This will be discussed in more detail in Section 4.4.

At the beginning of the time step, it is crucial to ensure that the particles are in valid initial positions $\boldsymbol{x}_i$. This step is necessary because, in some cases, convergence may not have been fully achieved in the previous time step, or particles may have been displaced to invalid positions due to external influences (e.g., user-modified collision objects). If left uncorrected, these invalid positions can introduce unwanted kinetic energy into the system, leading to non-physical behavior. The smaller the time step, the more pronounced these artifacts become. Any remaining penetrations in the initial positions can be resolved by performing stabilization using collision constraints, as suggested in [MMC+14]. The resulting positional corrections are applied to the initial and predicted positions, stabilizing the entire system and reducing the likelihood of introducing erroneous energy. Typically, one or two stabilization iterations are sufficient. Although this pre-stabilization may not guarantee perfect convergence, it minimizes visual artifacts and enhances the overall stability of the simulation.

Subsequently, all constraints are projected onto the predicted positions. First, collision and friction constraints are applied to all collision pairs. Then, a rigid transformation is determined for each rigid body using shape matching, which yields the current target positions for the particle representation of the rigid body when applied to the initial shape. These target positions are then enforced through a positional constraint. The total positional correction for each particle is accumulated from all resulting positional corrections and under-relaxed by dividing by the number of involved constraints, as described in Section 4.3.2, before being applied to the predicted positions. Five to ten solver iterations are typically required to achieve a convergent system. In the final step, the resulting velocities are calculated (see Equation 4.6), and the final positions $\boldsymbol{x}_i$ are updated from the final predicted positions $\boldsymbol{x}_i^*$.

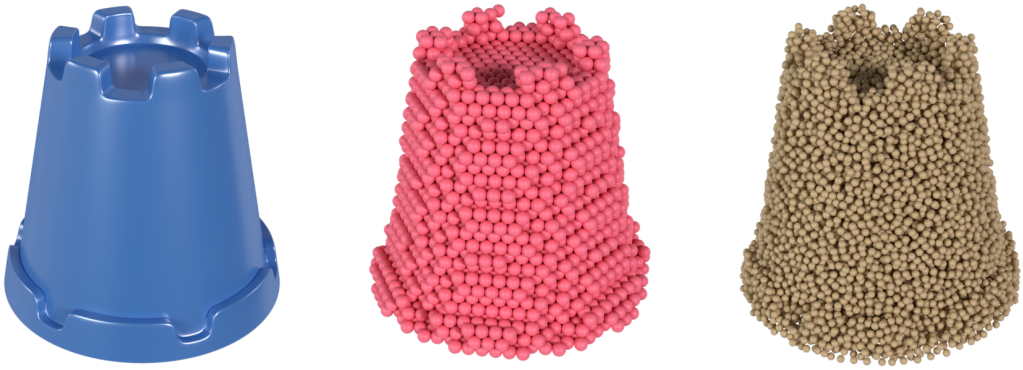
Each of the *forall* loops in Algorithm 4 can be executed in parallel. Given the typically moderate number of rigid bodies, it is advisable to execute the rigid body loop in line 17 sequentially while parallelizing the inner particle loop in line 19. It should be noted that a particle may have multiple contact partners, which can lead to race conditions when updating the resulting non-penetration and friction positional corrections. Therefore, atomic operations must be used during these updates to ensure consistency.

4.4 | High-resolution Simulation

The previous section described how granular materials can be simulated using position-based constraints, considering all contacts and collisions between individual particles. Although this method allows for efficient parallelization, it is still not feasible to simulate the large particle numbers required for most fine-grained granular materials in real time. One approach to significantly increase the number of particles without adding too much computational complexity involves running a second, higher-resolution simulation based on the results of the first simulation, as proposed in [ATO09] and [IWT12]. The general idea is that the new high-resolution particles are simulated without collision detection and constraint solving, depending solely on the low-resolution particles from the first simulation.

4.4.1 | Motivation

Depending on the grain size, hundreds of thousands to millions of particles are required for a realistic representation of fine-grained granular materials. Calculating the interactions between all these particles—including collisions and friction—requires vast computational resources. Furthermore, the required time steps become smaller as the particle size decreases, making real-time simulation even more difficult to achieve. However, the macroscopic behavior of granular materials depends less on the grain size itself and more on the properties of the individual grains [BKM+07]. Therefore, the low-resolution particles can be used to characterize the general macroscopic behavior of the granular material. Newly introduced high-resolution particles serve to improve the visual and structural detail of the simulation without altering the overall dynamics established by the low-resolution simulation. Properties (e.g., velocity) can be transferred to the high-resolution particles through interpolation from the surrounding low-resolution particles to ensure that these smaller particles follow the macroscopic flow of the low-resolution simulation. However, it is also necessary to allow external forces to act on these particles to preserve the individuality of certain high-resolution particles, particularly at the surfaces or when they are not within the granular flow. Describing the motion of high-resolution particles in this way eliminates the need to calculate internal forces within the granular material. In the position-based context, this means that positional corrections for high-resolution particles can be omitted, as well as collision detection between them. This allows for larger time steps and a temporal decoupling between the low-resolution and the high-resolution simulation.



(a) (b) (c)

Figure 4.4: The mesh of a sandcastle tower mold in (a) is volume-sampled using an HCP pattern in (b) with low-resolution particles and filled in (c) through randomized volume sampling with high-resolution particles of a smaller radius.

4.4.2 | Initial State

Each low-resolution particle occupies a larger portion of the volume within the granular material compared to the smaller high-resolution particles. A key factor for achieving visually pleasing simulation results is the initial arrangement of these particles at the beginning of the simulation. First, the initial volume of the granular material is filled with low-resolution particles, as described in Section 4.3. These particles serve as guide particles and not for visualization purposes. They should be densely packed using the HCP lattice sampling method described in Section 3.4.2 to prevent volume loss of the granular material. For the high-resolution simulation, the same volume is filled with high-resolution particles. It is important to emphasize that these particles do not share the volume with the low-resolution particles. Instead, they can fully occupy the space as if it were empty. Filling the volume using the randomized sampling method from Section 3.4.1 leads to more irregular and realistic effects. Since these particles follow the low-resolution particles, there is no collapse of the material, as would be the case if the low-resolution particles were not densely sampled. Other approaches include filling only the volume of low-resolution particles [ATO09] or uniformly distributed sampling around the center of the low-resolution particles [IWT12].

Consequently, the granular material comprises N_{hr} high-resolution particles with radius r_{hr}. To avoid confusion with low-resolution particles, attributes of high-resolution particles are indicated by a plus sign in the superscript (e.g., the position $\boldsymbol{x}_i^+$ of a high-resolution particle with index i) while low-resolution particles are denoted without

the superscript. Figure 4.4 shows an example of an initial volume in (a) filled with low-resolution particles in (b) and high-resolution particles in (c).

4.4.3 | Method

The equations of motion for the high-resolution particles can be derived based on the principles defined in Section 4.4.1. As in Section 4.3, Newton's second law of motion is the foundation. Similarly, the position

$$\boldsymbol{x}_i^+(t_n + \Delta t_{hr}) = \boldsymbol{x}_i^+(t_n) + \Delta t_{hr} \boldsymbol{v}_i^+(t_n + \Delta t_{hr}) \tag{4.39}$$

of a high-resolution particle with index i is determined through symplectic Euler integration for a given time step $t_n \rightarrow t_n + \Delta t_{hr}$. It is important to recall that larger time step sizes Δt_{hr} can be used for the high-resolution particles since no collision detection is needed, and $\Delta t_{lr} \neq \Delta t_{hr}$. The remaining unknown variable is the velocity $\boldsymbol{v}_i^+$ of the particle at the new time $t_n + \Delta t_{hr}$. As discussed in Section 4.4.1, this velocity should be interpolated from the velocities of the surrounding low-resolution particles for a particle within the granular flow. Before simulating the high-resolution time step $t_n \rightarrow t_n + \Delta t_{hr}$, all needed low-resolution time steps Δt_{lr} from t_n up to $t_n + \Delta t_{hr}$ should have been simulated (see Algorithm 5). Therefore, the velocities of the low-resolution particles at $t_n + \Delta t_{hr}$ are known.

In SPH [GM77], physical field quantities (e.g., velocity) can be interpolated at any point in space from discrete particle data by weighting contributions from neighboring particles using kernel functions. This approach inspires the interpolation of the velocities at the positions of the high-resolution particles. For each neighboring low-resolution particle with index j around a high-resolution particle with index i, the velocity contribution to the interpolation is weighted according to a kernel function W_{hr}, which depends on the distance $|\boldsymbol{x}_{ij}| = |\boldsymbol{x}_j - \boldsymbol{x}_i^+|$ between the two particles and the smoothing length h_{hr}. The interpolated velocity

$$\boldsymbol{v}_{a,i}^+(t_n + \Delta t_{hr}) = \frac{1}{\sum_j W_{hr}(|\boldsymbol{x}_{ij}|, h_{hr})} \sum_j W_{hr}(|\boldsymbol{x}_{ij}|, h_{hr})\ \boldsymbol{v}_j(t_n + \Delta t_{hr}) \tag{4.40}$$

must also be normalized by the sum of all involved kernel weights, so that the total influence of the surrounding particles sums to unity, ensuring physical consistency in velocity. The *smoothing length* is a scaling parameter that controls the influence of the kernel function, determining how distant neighbors affect the velocity of the high-resolution particle. In modern SPH implementations, the smoothing length is often

variable to respond to local particle densities dynamically [FDT+09]. However, it is fixed here at $h_{hr} = 3r_{lr}$. In addition to the smoothing length, the *support radius* defines the maximum distance around a particle within the kernel function, which will have a non-zero influence.

Several kernel functions are used in SPH, each with different characteristics. The Gaussian kernel [M92] is very smooth, but its support radius extends across the entire domain, making it computationally inefficient for large numbers of particles. Other popular kernel functions, such as the cubic spline kernel [R67] and the quintic spline kernel [S46], have shorter support radii and are, therefore, better suited for simulations with large particle numbers. However, for the high-resolution simulation, another kernel function

$$W_{hr}(|\boldsymbol{x}_{ij}|, h_{hr}) = \max\left(0, \left(1 - \frac{|\boldsymbol{x}_{ij}|^2}{h_{hr}^2}\right)^3\right), \tag{4.41}$$

initially used for surface reconstruction from particle data [ZB05], has proven effective [IWT12]. This kernel is well-shaped, decays smoothly to zero, and is particularly efficient to compute as it avoids using square roots since $|\boldsymbol{x}_{ij}|^2 = \boldsymbol{x}_{ij}^\mathsf{T}\boldsymbol{x}_{ij}$. Additionally, its support radius equals the smoothing length. Consequently, a search radius 10–20 % larger than the smoothing length for neighborhood search in the low-resolution simulation is sufficient to reuse it for the high-resolution part. For further information on different kernel functions in SPH, refer to [LL10].

With the interpolated velocity $\boldsymbol{v}_{a,i}^+$, a high-resolution particle with index i follows the macroscopic flow of the granular material. As noted in Section 4.4.1, particles near the surface of the flow or outside it should be able to move freely to mimic the realistic behavior of individual high-resolution particles. The velocity

$$\boldsymbol{v}_{f,i}^+(t_n + \Delta t_{hr}) = \boldsymbol{v}_i^+(t_n) + \Delta t_{hr}\frac{\boldsymbol{f}_{\text{ext}}}{m_i^+} \tag{4.42}$$

of a free motion independent of the granular flow can be calculated analogously to the predicted velocity of the low-resolution particles (see Equation (4.5)). Again, the only external force $\boldsymbol{f}_{\text{ext}}$ acting here is Earth's gravitational force $\boldsymbol{f}_g$, which leads to an acceleration $\boldsymbol{g}_e = \boldsymbol{f}_g/m_i^+ \approx -9.81\,\text{m}\,\text{s}^{-2}\cdot\hat{\boldsymbol{e}}_2$ independent of the mass m_i^+ of the high-resolution particle.

When should a high-resolution particle follow the granular flow, and when should it move freely? In [ATO09], a discrete distinction is made: if there is more than one low-resolution particle within the support radius, the high-resolution particle should

follow the flow; otherwise, it should move freely. A smoother, more continuous approach involves a gradual transition, as proposed in [IWT12]. Particles within the granular material strictly follow the flow, while those near the surface partially follow the flow, with external forces blended in. Particles located entirely outside the material move completely freely. The resulting velocity

$$\boldsymbol{v}_i^+(t_n + \Delta t_{hr}) = (1 - \lambda_i^+)\boldsymbol{v}_{a,i}^+(t_n + \Delta t_{hr}) + \lambda_i^+ \, \boldsymbol{v}_{f,i}^+(t_n + \Delta t_{hr}) \tag{4.43}$$

of a high-resolution particle with index i can be formulated using a linear blending parameter $\lambda_i^+ \in [0, 1]$. Within the granular flow, where only the interpolated velocity should be applied, $\lambda_i^+ = 0$ must hold. For a high-resolution particle with no low-resolution neighbors within its support radius, $\lambda_i^+ = 1$ must hold.

The influence on a high-resolution particle in a sparse neighborhood can be defined by its nearest neighbor. Since $0 \leq W_{hr} \leq 1$ holds, one approach is to set $\lambda_i^+ = 1 - \max_j W_{hr}(|\boldsymbol{x}_{ij}|)$ in this case. A high-resolution particle is considered in a sparse neighborhood if no other particle is within a radius r_{lr}, which is equivalent to $\max_j W_{hr}(|\boldsymbol{x}_{ij}|) \leq W_{hr}(r_{lr})$. Even if a few low-resolution particles are close, the given high-resolution particle may still be in a sparse region. The larger the quotient $\max_j W_{hr}(|\boldsymbol{x}_{ij}|) / \sum_j W_{hr}(|\boldsymbol{x}_{ij}|)$, the sparser the region. Empirical tests suggest that a reasonable threshold for this quotient is 0.6. Therefore,

$$\lambda_i^+ = \begin{cases} 1 - \max_j W_{hr}(|\boldsymbol{x}_{ij}|) & \text{if} \quad \max_j W_{hr}(|\boldsymbol{x}_{ij}|) \leq W_{hr}(r_{lr}) \text{ or } \frac{\max_j W_{hr}(|\boldsymbol{x}_{ij}|)}{\sum_j W_{hr}(|\boldsymbol{x}_{ij}|)} \geq 0.6 \\ 0 & \text{else} \end{cases} \tag{4.44}$$

can be used.

4.5 | Implementation

The low-resolution simulation from Section 4.3 and the high-resolution simulation from Section 4.4 can be merged to describe the complete simulation loop. Since the low-resolution simulation requires significantly smaller time step sizes Δt_{lr} than typical display refresh rates, it is advisable to define a target time step size Δt_{tar} up to which simulation steps should be accumulated. For all simulation scenarios in this thesis, a desired frame rate of 50 Hz is used, corresponding to a target time step size $\Delta t_{\text{tar}} = 20\,\text{ms}$. Algorithm 5 presents a complete simulation step for a target time step in pseudocode, where $\Delta t_{\text{tar}} > \Delta t_{hr} > \Delta t_{lr}$.

Algorithm 5: A full simulation step for a target time step Δt_{tar}

Data: Positions $\boldsymbol{x}_i$, $\boldsymbol{x}_i^+$ and velocities $\boldsymbol{v}_i$ at a time t_n
Result: Positions $\boldsymbol{x}_i$, $\boldsymbol{x}_i^+$ and velocities $\boldsymbol{v}_i$ at a time $t_n + \Delta t_{\text{tar}}$

```
t_it,hr ← 0
reset Δt_hr
while t_it,hr < Δt_tar do
    if t_it,hr + Δt_hr > Δt_tar then
        Δt_hr ← Δt_tar − t_it,hr
    t_it,hr ← t_it,hr + Δt_hr
    t_it,lr ← 0
    reset Δt_lr
    while t_it,lr < Δt_hr do
        if t_it,lr + Δt_lr > Δt_hr then
            Δt_lr ← Δt_hr − t_it,lr
        t_it,lr ← t_it,lr + Δt_lr
        x_i, v_i ← low-resolution simulation step(Δt_lr)
    x_i^+ ← high-resolution simulation step(x_i, v_i, Δt_hr)
```

When selecting an appropriate time step size for the low-resolution simulation, ensuring that collisions between particles can be resolved accurately is crucial. Specifically, the time step must be small enough to prevent particles from *tunneling* through each other. Tunneling would occur if their displacement during a single time step exceeded their combined radii. Accordingly, the time step size is limited by the particle radius and the velocity at which the particles move. To address this, the Courant-Friedrichs-Lewy (CFL) condition [LFC28]

$$\Delta t_{lr} = \frac{C_{\text{crt}} \cdot r_{lr}}{v_{\max}} \tag{4.45}$$

can be applied, where r_{lr} is the low-resolution particle radius, $v_{\max}$ is the magnitude of the maximum particle velocity, and C_{crt} is the Courant number, which controls the conservativeness of the time step selection. For the simulation results in this chapter, the Courant number is $C_{\text{crt}} = \frac{1}{2}$. This ensures that low-resolution particles do not move more than half their radius in a single low-resolution time step, preventing tunneling while allowing reasonably large time steps. The CFL condition typically results in a dynamically adapting time step size as the maximum velocity fluctuates throughout the simulation. However, for the results in Section 4.6, the simulation is initially run with a smaller time step to determine the maximum velocity across the entire sequence. A static time step size Δt_{lr} is established based on this. This fixed time step is then used for the simulation results and timings.

The time step size Δt_{hr} for the high-resolution simulation is less restrictive, as no collision handling is required. Theoretically, the time step size Δt_{hr} could equal the target time step Δt_{tar}. However, in practice, choosing a smaller time step size is advisable, as excessively large time steps can lead to artifacts (e.g., high-resolution particles not reaching boundaries closely enough). Conversely, very small time steps result in more noticeable clumping of high-resolution particles and increased computation time. For the results in the next section, a time step of $\Delta t_{hr} = \frac{2}{5}\Delta t_{\text{tar}}$ was used.

The simulation was implemented using the Taichi programming language [HLA+19], designed for high-performance computing on massively parallel devices (e.g., GPUs) to leverage the efficient parallelizability of the simulation loop.

4.6 | Results

The following five simulation scenarios illustrate the quality and performance of the two-stage granular simulation.

In the first scenario—*Squirrel*—a squirrel made of granular material is dropped onto the ground. The low-resolution simulation comprises 14K particles and has a time step of $\Delta t_{lr} = 1.4\,\text{ms}$. As in all scenarios in this section, the high-resolution simulation has a time step $\Delta t_{hr} = 8\,\text{ms}$. It features 227K particles representing the squirrel. The static friction coefficient is $\mu_s = 0.6$, and the kinetic friction coefficient is $\mu_k = 0.55$. Figure 4.5 shows this scenario.

The second scenario—*Bucket*—involves an excavator's bucket moving through a pile of granular material. The pile comprises 18K low-resolution particles and 288K high-resolution particles. The time step is $\Delta t_{lr} = 1.6\,\text{ms}$, with the same friction coefficients as in the previous example. Figure 4.6 illustrates this scenario.

The *Hourglass* scenario simulates a large-scale hourglass. The granular material inside the hourglass comprises 19K low-resolution particles and 411K high-resolution particles. The surface of the hourglass, which defines the domain boundary for the low-resolution simulation, consists of an additional 18K static particles. The low-resolution time step is $\Delta t_{lr} = 1\,\text{ms}$, and the friction coefficients are $\mu_s = 0.4$ and $\mu_k = 0.45$. Figure 4.7 shows this scenario.

The *Slope slide* scenario features the collapse of a tower made of granular material, which then slides down a sloped surface. The tower is simulated with 38K low-resolution

Figure 4.5: Simulation scenario *Squirrel*

Figure 4.6: Simulation scenario *Bucket*

Figure 4.7: Simulation scenario *Hourglass*

particles and 597K high-resolution particles. The time step is again $\Delta t_{lr} = 1\,\text{ms}$ and friction coefficients are $\mu_s = 0.5$ and $\mu_k = 0.45$. Figure 4.8 presents this scenario.

In the final scenario—*Armadillo*—granular material is continuously poured from a silo over a static object in the shape of an armadillo. The armadillo's surface is sampled with 3K particles, while the granular material consists of 20K low-resolution and 652K high-resolution particles. The time step is $\Delta t_{lr} = 1\,\text{ms}$, with friction coefficients matching those of the *Squirrel* scenario. Figure 4.9 illustrates this scenario.

Table 4.2 presents the average runtimes for a single simulation frame for the different scenarios. All scenarios were simulated on a system equipped with an Intel Core i9-

Figure 4.8: Simulation scenario *Slope slide*

Figure 4.9: Simulation scenario *Armadillo*

Scenario	**Particles**		**Time/Frame**		
	lr	**hr**	**lr**	**hr**	**total**
(Fig. 4.5) Squirrel	14K	227K	10.5 ms	2.1 ms	12.6 ms
(Fig. 4.6) Bucket	18K	288K	16.3 ms	2.5 ms	18.8 ms
(Fig. 4.7) Hourglass	19K	411K	14.2 ms	4.3 ms	18.5 ms
(Fig. 4.8) Slope slide	38K	597K	28.8 ms	6.8 ms	35.6 ms
(Fig. 4.9) Armadillo	20K	652K	14.5 ms	6.2 ms	20.7 ms

Table 4.2: Performance analysis of the individual simulation scenarios.

13900H CPU and an Nvidia GeForce RTX 4060 mobile GPU. Only the time to calculate the simulation without visualization was considered. For each scenario, the target frame rate is 50 Hz, which results in a target time step size $\Delta t_{\text{tar}} = 20$ ms. Therefore, runtimes below 20 ms represent real-time performance. The accumulated runtimes for the required low-resolution (lr) and high-resolution (hr) simulation steps are also listed separately.

Direct comparisons with state-of-the-art methods are challenging as the lack of publicly available code prevents testing these methods under identical scenarios. The chosen scenarios are intended to represent similar scenes to provide a degree of comparability. However, the results focus on particle numbers that allow for real-time simulation, which differs in part from the particle numbers used in the compared scenes. The performance evaluations of other methods are based on data provided in the respective publications. It is important to note that some of these simulations were timed on older hardware, and some publications lack details on frame rates or time step sizes. Despite these limitations, all the compared methods are slower by two or more orders of magnitude. They are far from real-time capable, highlighting the efficiency of the approach presented in this chapter.

In [ATO09], a scenario similar to Figure 4.9 is presented, featuring 40K guide particles and 1.6M visualization particles. The computation time for a 25 s video is reported to be 69 hours at a frame rate of 30 Hz, resulting in a time per frame of 3.3×10^5 ms. Additionally, a smaller scenario, comparable to Figure 4.5, with 2.5K guide particles and 50K visualization particles, required 1 hour and 25 minutes to compute a 4.5 s video, resulting in a time per frame of 3.8×10^4 ms (based on an assumed 30 frames per second).

In [NGL10], an hourglass scenario (similar to Figure 4.7) uses 44K simulation particles, which are increased to 2.4M visualization particles per frame when rendered. The reported time per frame is 6.5×10^3 ms, although no details on the time step size or target frames per second are given. For a better comparison with the approach presented in this chapter, the hourglass scenario from Figure 4.7 was repeated with larger particle counts, where real-time performance was no longer achievable. For 55K low-resolution particles and 2.4M high-resolution particles, a total computation time of 100 ms per frame was required, with 89 ms for the low-resolution simulation and 11 ms for the high-resolution stage.

In [IWT12], a bulldozer scenario similar to the scenario in Figure 4.6 simulates 38K low-resolution particles and 1.4M high-resolution particles with a time per frame of 2.5×10^3 ms, again without specifying the target frame rate.

In [AO11], a scenario similar to the one in Figure 4.5 features 50K particles and a time per frame of 5.9×10^3 ms. Another scenario with 20K particles and a time per frame of 3.4×10^3 ms is presented further. Both use a target frame rate of 24 Hz.

In [KGP+16], another hourglass scenario is presented, simulating 460K particles at 1.2×10^5 ms per frame with a target frame rate of 48 Hz. Additionally, a scenario with 795K particles takes 4.1×10^5 ms per frame at 72 Hz. A scenario like the one in Figure 4.7, but with an order of magnitude more particles (1.96M), requires 1.5×10^6 ms per frame at a target frame rate of 24 Hz.

In [HFG+18], 1.0M falling particles are simulated interacting with rigid paddles at an unspecified target frame rate with 2.0×10^4 ms per frame.

Finally, in [ZCZ+19], a scenario comparable to the one in Figure 4.8 is given, simulating 786K particles on the GPU with a time per step of 290 ms and a time step size of 0.5 ms, resulting in 1.16×10^4 ms per frame at a target frame rate of 50 Hz.

Overall, all other methods are far from achieving real-time performance. However, it is important to note that the latter three methods accurately simulate all particles.

5

Lighting Estimation

Providing the user with an immersive experience is critical to the success of an AR application. This includes the seamless and realistic integration of virtual objects into a real scene through photometric registration. Therefore, it is necessary to capture the real environment's lighting situation accurately. Lighting estimation helps mimic real-world lighting conditions, allowing virtual elements to have realistic shadows, reflections, and highlights. In addition to ensuring visual consistency, lighting estimation is important in providing spatial awareness to the user. Accurately illuminating virtual objects enables better judgments of distance, size, and orientation. Figure 5.1 illustrates the importance of accurate illumination. In (a), a constant illumination results in only the object's outlines being visible due to the absence of shadows and highlights. In (b), arbitrary lighting is used to illuminate the object, where the light direction, color, and intensity do not match the existing lighting conditions of the real scene in the background. While this approach makes the three-dimensionality of the object visible, it fails to integrate seamlessly into the real scene. In contrast, in (c), an illumination is applied, where the light sources and ambient lighting correspond to the actual lighting conditions of the real background scene. As a result, the object integrates more naturally into the existing scene.

(a) Constant illumination (b) Not matching illumination (c) Matching illumination

Figure 5.1: A virtual squirrel is inserted into the real scene. In (a), the object is illuminated with a constant ambient color. In (b), the illumination of the object does not match the real scene, while in (c) it does.

5.1 | Outline

This chapter presents a parametric lighting estimation method, determining the lighting conditions of a real scene based on a single RGB camera image. The method is specifically designed for AR applications and was initially presented in [SSS23]. Section 5.2 discusses different works that paved the way for lighting estimation methods, followed by an introduction to current methods related to the presented one. Section 5.3 introduces the parametric lighting representation, describing the lighting situation using geometric and photometric parameters. Section 5.4 shows how a suitable dataset can be generated from panoramic images, allowing a neural network to be trained to estimate these parameters for a given input image. Section 5.5 discusses the architectural choices for this neural network and considerations for its training. The chapter concludes in Section 5.6 with a comprehensive evaluation of the presented method and a discussion of its advantages and limitations.

5.2 | Related Work and Background

The digital camera is the first thing that comes to mind when digitally capturing a real scene (recall Section 2.2). The image captured by a digital camera is an array of brightness values (recall Section 2.1.2). These brightness values depend on the exposure—the product of irradiance and exposure time—at each camera sensor pixel. The camera response function transforms exposure into digital brightness values, referred to as *intensity*. Intensity values are recorded separately for each color channel and are typically quantized to 8 bit precision, resulting in 256 discrete intensity levels per channel. Due to this limited number of levels, scenes with high contrast differences cannot be accurately or losslessly represented, which is why such images are referred to as low dynamic range (LDR) images. The camera response function applies a nonlinear mapping between exposure and intensity to approximate the typically high-contrast scenes of the real world within the constraints of LDR images. This mapping compresses the dynamic range into 8 bit values approximating human visual perception. However, due to this nonlinearity, a region that appears twice as bright in the image does not necessarily correspond to twice the physical radiance in reality. For more details on how images are captured and represented digitally, refer to [GW18].

In [DM97] an algorithm is proposed whereby multiple LDR images of the same scene, captured with varying exposure times, can be fused to restore the original radiance of each pixel up to a scalar factor. Therefore, the resulting high dynamic range (HDR)

radiance map has a linear relationship to the actual physical radiance values in the scene. Throughout this thesis, HDR images or HDR maps are abbreviated terms for HDR radiance maps.

In [D98], such HDR maps are used to realistically render virtual objects into images of real scenes. For this purpose, a reflective metal sphere—the *light probe*—is placed in the real scene at the insertion point of the virtual object. The reflection from the metal sphere enables the representation of the physical lighting conditions in the vicinity by approximating a 360° radiance map, also known as an *environment map*. These environment maps can be used for IBL or environment sampling (recall Section 2.3.3).

Instead of determining radiance maps, other works focus on separately estimating illumination and reflection for inserted virtual objects [LN12; BM15; LN16]. Other methods concentrate on identifying individual light sources in the scene and characterizing them through parameters for virtual light sources (recall Section 2.3.2). The direction of a single light source can be estimated by the surface [P82; BH85], contour [NE01], or texture [KP03; VZ04] of a reference object. Multiple light sources can even be identified using a reference object of known geometry [GHH01; PSP09] or by approximating the surface normals of shiny objects [LGH+13]. All these classical approaches have in common that they require multiple images, reference objects in the images, or a more detailed understanding of the underlying scene geometry.

Deep learning-based approaches reduce the needed amount of scene information but, in return, require datasets, as described in Section 2.4. Given the substantial differences in lighting conditions between indoor and outdoor scenes with the sun as the primary light source, most works specialize in either indoor or outdoor scenes with corresponding datasets. In [HSH+17], a deep neural network extracts the parameters of a physically-based sky model from a single outdoor camera image. This model allows for the estimation of an outdoor HDR environment map. Indoor scenes pose a more challenging problem due to multiple light sources. In [GSY+17], this is addressed using a two-stage approach. In the first stage, light directions are predicted by a neural network based on an input image. Later, these directions are transformed into a 360° HDR map by a second network. Similarly, in [SF19], a multi-stage approach is proposed where an LDR 360° image is initially predicted from an input image using one network and then transformed into an HDR environment map by a second network. In [SMT+20], a volumetric RGBA model of the entire scene is created from a stereoscopic image pair. This model allows the creation of a light probe at any point in the scene. A 360° HDR environment map can be directly predicted in a single-stage approach from an input

image captured by a conventional camera using a neural network, as shown in [SK21; WYL+22]. The method in [SK21] is subsequently optimized for real-time AR applications on mobile devices.

Similar to classical methods, there are also learning-based methods that do not predict environment maps but instead parameters of lighting representations. Some approaches represent spatially varying lighting through spherical harmonic coefficients [CSC+18; GSH+19] or spherical Gaussians [ZZY+21; ZYZ+22]. These are a numerical approximation of the environment map through a series expansion with spherical basis functions (recall Section 2.3.3). In [GHS+19], a neural network is trained to identify parametric light sources from a single input image. The method described in [GHS+19] is most closely related to the one presented in this chapter. The approach in [GHS+19] uses HDR maps to create the lighting parameters of the training data. The light source parameters are determined by fitting ellipses to the brightest regions of the HDR maps. The position, size, and average color value of the ellipse yield the according parameters of the light source. In addition, a second network is used to estimate depth maps for the HDR maps, which define the distance of the light source. In [DEH+23], a hybrid method that predicts an LDR environment map and extends it to HDR image depth using spherical Gaussians is introduced. This method is suitable for both indoor and outdoor scenes.

5.3 | Lighting Representation

As mentioned previously, different lighting estimation methods adopt different representations for their results. Here, a parametric description of the primary light source and remaining illumination is used. A set of parameters

$$\left(\hat{\boldsymbol{d}}_l, \boldsymbol{c}_l, \boldsymbol{a}_l, o_l\right) \tag{5.1}$$

describes the lighting situation: a light direction $\hat{\boldsymbol{d}}_l$, its light color $\boldsymbol{c}_l$, an ambient color $\boldsymbol{a}_l$, and the so-called *contrast level* o_l. The light direction $\hat{\boldsymbol{d}}_l$ is the direction of the primary light source as a Cartesian unit vector relative to the input image plane. The light color $\boldsymbol{c}_l$ is an RGB value with individual channel values from 0 to 1. The ambient color $\boldsymbol{a}_l$ describes the color of the overall, uniform illumination in the absence of the primary light source. It establishes a foundational color atmosphere within a scene and provides general visibility by lighting areas not directly illuminated by the primary light source. It is another RGB value with channels from 0 to 1. The contrast level o_l is a scalar value between 0 and 1. It characterizes the intensity ratio between the primary light source and the remaining ambient illumination of the scene. A theoretical value of $o_l = 1$ indicates a

(a) o_l close to 1 (b) o_l close to 0

Figure 5.2: A virtual squirrel is inserted into a scene with high contrast on a sunny day, resembling o_l close to one in (a), and a scene with low contrast on an overcast day, resembling o_l close to zero in (b).

complete absence of secondary light sources. In such scenarios, the resulting shadows of an object are pitch black, as they receive no illumination from any direct or indirect light source. Scenes with o_l closer to 1 resemble direct sunlight or a focused spotlight and are characterized by high contrast (see Figure 5.2a). Conversely, a theoretical value of $o_l = 0$ indicates completely uniform illumination from all directions, resulting in the absence of any contrast. Scenes with o_l closer to 0 resemble overcast skies or highly diffuse indoor lighting (see Figure 5.2b). As a result, cast shadows become progressively weaker until they disappear entirely at $o_l = 0$. In Chapter 6, the contrast level will be used to control the rendering opacity of shadow textures.

5.4 | Dataset

For a data-driven approach, a dataset containing suitable training data is needed to train the neural network. The lighting estimation method aims to predict the lighting parameters described in Section 5.3 for a given input image. Consequently, the dataset must comprise tuples of input images and corresponding lighting parameters (Equation (5.1)). For many common problems, there are already suitable datasets on which different network architectures can be trained and compared. Notable examples in image classification include MNIST [LBB+98; D12], CIFAR-10 [K09], CIFAR-100 [K09], and ImageNet [DDS+09]. However, no such dataset exists for the lighting parameters described in Section 5.3. A suitable dataset can be composed by using a sufficiently large number of scenes with known lighting conditions for which matching input images can be found.

5.4.1 | Environment Maps

HDR environment maps are suitable for describing the lighting conditions in a scene. The floating-point intensity values of individual pixels are proportional to the physical quantity of radiance (recall Section 5.2). Since brightness variations in such 360° HDR images have a physical meaning, individual lighting parameters can be extracted from them. However, a large number of environment maps are required to acquire a sufficient amount of training data.

The Laval Indoor HDR Dataset[1] [GSY+17] is such a large collection of environment maps comprising 2100 panoramas of indoor environments. Each panorama is high-resolution with 7768×3884 pixels. The dataset was captured using a Canon 5D Mark III camera with a Sigma 8 mm fish-eye lens mounted on a tripod. For each panorama, a series of seven shots with different exposures—called *exposure bracket* in photography—was taken at 60° intervals. For such an exposure bracket, the aperture is kept constant while the exposure time is varied to maintain a consistent depth of field. The *ISO* value can be increased to compensate if a longer exposure time is not practical due to the absence of a tripod or in the case of moving objects. Increasing the ISO will result in additional noise. The first image in the exposure bracket is typically a neutral exposed image. For the series, exposure is adjusted in increments of relative exposure value (EV). An increase of 1 EV halves the amount of light reaching the sensor during exposure. In [GSY+17], a dynamic range of 22 f–stops is specified, corresponding to 9 EV and, therefore, a bracketing of 1.5 EV for the seven exposures. A total of 42 images were captured per panorama: seven exposures for each of the six shooting angles at 60° intervals. Maintaining a constant, absolute EV for the first image of the exposure series across all shooting angles is recommended to ensure consistency in the final panorama.

Several panoramic projections exist for mapping a 3D scene onto a flat surface. The images in this dataset are transformed using the equidistant cylindrical projection, also known as *equirectangular projection* [S87, pp. 90–91]. This projection is typically used for environment maps in computer graphics. It is a latitude/longitude projection of the panoramic sphere onto an image with an aspect ratio of 2 to 1. Horizontal and vertical lines are projected as straight lines, while all other lines become curved. Other commonly used projections include Mercator [S87, pp. 38–47], Vedutismo [SPG10], and stereographic projection [S87, pp. 154–163].

For the Laval Indoor HDR Dataset, the 42 images are stitched into an HDR panorama

[1] The dataset is publicly available at: `http://www.hdrdb.com/indoor/`

Figure 5.3: A selection of four equirectangular panoramas from the Laval Indoor HDR Dataset presented as LDR images.

using the commercial software PTGui Pro[2]. For more information on panorama stitching, refer to [SS97; BL07] since the manufacturer provides no information about the algorithms used in the software. Figure 5.3 shows four panoramas from the dataset. Image regions obstructed by the tripod are masked in black in the original dataset (see Figure 5.3). In the following, for practical reasons, the black pixels are replaced by copying the last vertically adjacent non-black pixels. It should also be noted that the Figure does not reflect the images in the actual dynamic range of the HDR images because they cannot be displayed on a conventional monitor or made visible through printing. It is a tone-mapped LDR version of the image; for more details, see Section 5.4.2.

5.4.2 | Input Images

For the training data, input images with corresponding lighting parameters are required. The lighting parameters are determined from the HDR panoramas (see Section 5.4.3). Therefore, the input images should be excerpts from the panoramas. Since the network is designed to receive regular camera images as input (e.g., from a AR device), they must have a similar dynamic range. Consequently, the HDR panorama must first be converted into an LDR panorama. HDR data is in a linear relationship with the actual radiance values of the scene. However, an LDR image captured with a regular camera

[2]The software is available at: `https://ptgui.com/`

produces a nonlinear representation of radiance to mimic a dynamic range closer to human perception (recall Section 5.2). Different *tone mapping* operators transform the radiance values of an HDR image into RGB values of an LDR image by emulating the response function of a camera sensor or human perception.

A simple and commonly used tone mapping operator is the Reinhard operator [RSS+02]. It uses luminance $L^{(v)}_{\text{HDR}}(u,v)$ (i.e., perceived brightness) instead of per pixel RGB radiance $\boldsymbol{L}^{(e)}_{RGB}(u,v) = (R_{uv}, G_{uv}, B_{uv})$. Therefore, the radiance values of individual color channels must initially be converted to the scalar luminance

$$L^{(v)}_{\text{HDR}}(u,v) = 0.2126 \cdot R_{uv} + 0.7152 \cdot G_{uv} + 0.0722 \cdot B_{uv}\,. \tag{5.2}$$

There are different standards for the individual weights assigned to each color channel. The values used here conform to the International Telecommunication Union (ITU) standard BT.709 [I15]. The Reinhard operator transforms pixel-wise HDR luminance $L^{(v)}_{\text{HDR}}(u,v)$ to pixel-wise LDR luminance

$$L^{(v)}_{\text{LDR}}(u,v) = \frac{L^{(v)}_{\text{HDR}}(u,v)\left(1 + \frac{L^{(v)}_{\text{HDR}}(u,v)}{(L^{(v)}_{\text{white}})^2}\right)}{1 + L^{(v)}_{\text{HDR}}(u,v)}\,. \tag{5.3}$$

$L^{(v)}_{\text{white}}$ represents the threshold luminance at and above which all values are mapped to pure white. Selecting $L^{(v)}_{\text{white}}$ as the maximum luminance is usually sufficient for most indoor scenes where the contrast between bright and dark areas is less pronounced than in outdoor scenes. New RGB color channel intensities for the LDR image $\boldsymbol{I}_{\text{LDR}}$ are obtained by scaling the RGB radiance values $\boldsymbol{L}^{(e)}_{RGB}(u,v)$ with the luminance $L^{(v)}_{\text{LDR}}(u,v)$. In the simplest case, this can be done by a linear scaling:

$$\boldsymbol{I}_{\text{LDR}}(u,v) = \boldsymbol{L}^{(e)}_{RGB}(u,v)\frac{L^{(v)}_{\text{LDR}}(u,v)}{L^{(v)}_{\text{HDR}}(u,v)}\,. \tag{5.4}$$

Other commonly used tone mapping operators include Drago [DMA+03], Durand [DD02], and Mantiuk [MKR+11], each with a different focus: Drago, as Reinhard, is a global operator that applies a logarithmic compression with a different base for each pixel, making it well suited for compressing very high dynamic ranges while preserving perceptual brightness. Durand, in contrast, is a local operator that enhances contrast by applying edge-preserving bilateral filtering, which helps preserve details without introducing halo artifacts around objects. Mantiuk operates on the luminance gradient of the input image, selectively smoothing out larger gradients while preserving fine details. For a more comprehensive overview of tone mapping, refer to [EMU17].

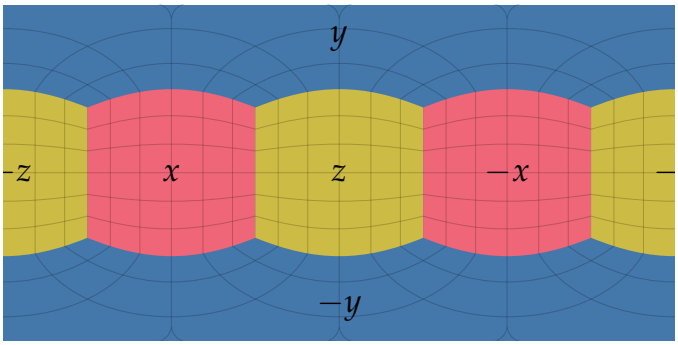
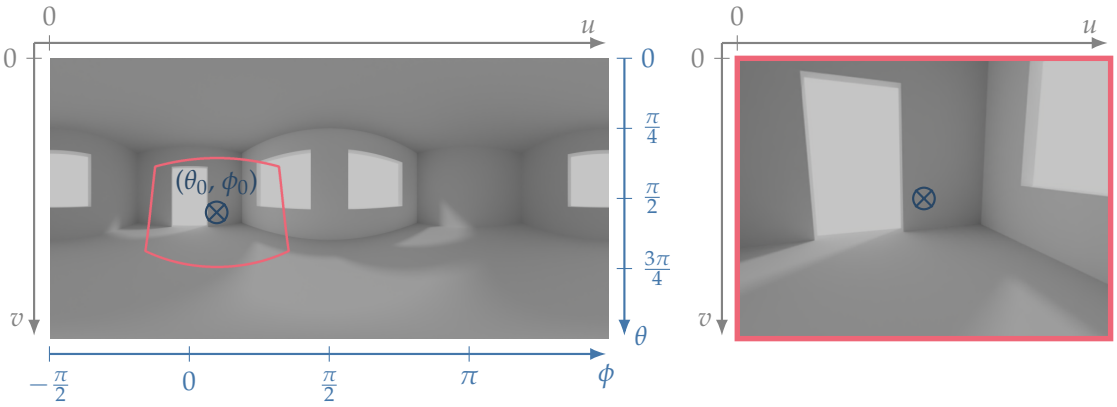


Figure 5.4: An equirectangular panorama from inside a unit cube, with each face of the cube aligned with a coordinate axis.

The resulting LDR panoramas are used to extract input images from them. It is important to note that the panoramas are presented in an equirectangular projection. Conventional camera images, on the other hand, represent the scene in a rectilinear projection. In rectilinear projection, all straight lines are represented as straight lines. This projection appears most natural to us, as our eyes and brain also interpret visual impulses with an approximately rectilinear mapping. However, it is unsuitable for representing panoramas with a larger viewing angle, as distortions become significant beyond a viewing angle of 120°. Panoramas with viewing angles exceeding 180° cannot be presented in rectilinear projection. Therefore, a section from the equirectangular panorama must be transformed into a rectilinear projection to obtain a suitable input image.

A rectilinear image is the result of an ideal perspective projection. Camera geometry (recall Section 2.2.1) describes mapping a 3D point to a 2D image coordinate. To create a perspective projection of a panorama, the panorama itself must be treated as 3D space, not an image. Therefore, a relationship between 3D points and panorama pixel coordinates must be defined. An equirectangular panorama is a 2D texture that can be mapped onto a unit sphere. Consequently, each pixel can be associated with unit spherical coordinates. However, there is no uniform definition of which region of the panorama corresponds to which direction in space. In the following, the convention is that the center of the panorama image corresponds to an orientation towards the positive z-axis. Figure 5.4 shows an equirectangular panorama from inside a unit cube, with each cube face labeled according to its corresponding coordinate axis, to visualize the convention. According to Equations (2.4) - (2.6), this results in unit spherical coordinates $\theta = \frac{\pi}{2}$ and $\phi = \frac{\pi}{2}$ for the center of the panorama (see Figure 5.5a). Each pixel $(u_p, v_p)^\intercal$ of the panorama, hereafter



(a) Equirectangular Panorama $\boldsymbol{I}_p$ (b) Rectilinear excerpt $\boldsymbol{I}_e$

Figure 5.5: An excerpt (red frame) from panorama $\boldsymbol{I}_p$ (a) is projected as a rectilinear image $\boldsymbol{I}_e$ (b). The crossed dot displays the view direction in angles (θ_0, ϕ_0).

referred to as image $\boldsymbol{I}_p$ with dimensions $w_p \times h_p$, corresponds to a location

$$\theta_p = \left(\frac{\pi v_p}{h_p}\right), \tag{5.5}$$

$$\phi_p = \left(\frac{2\pi u_p}{w_p}\right) - \frac{\pi}{2} \tag{5.6}$$

on the unit sphere and, therefore, to a Cartesian coordinate $\boldsymbol{x}_p$ following Equations (2.4) - (2.6), representing a point in the world coordinate system.

The section of the panorama used for projection is defined by the *view direction*, denoted as (θ_0, ϕ_0) (see crossed dot in Figure 5.5a). The view direction corresponds to the principal axis of the theoretical camera with which the rectilinear image excerpt $\boldsymbol{I}_e$ was captured. The camera and the world coordinate system have different orientations when $(\theta_0, \phi_0) \neq (\frac{\pi}{2}, \frac{\pi}{2})$ and are connected by a Euclidean transformation (recall Section 2.2.1). Both have the same origin, so the translation is $\boldsymbol{t}_c = \boldsymbol{0}$. The rotation $\boldsymbol{R}_c$ between them is defined by two rotation angles:

$$\Delta\theta = \theta_0 - \frac{\pi}{2}, \tag{5.7}$$

$$\Delta\phi = \phi_0 - \frac{\pi}{2}. \tag{5.8}$$

Therefore, rotation $\boldsymbol{R}_c$ between world and camera coordinates can be described as a composition of two rotations:

$$\boldsymbol{R}_c = \boldsymbol{R}_\phi \boldsymbol{R}_\theta. \tag{5.9}$$

A rotation $\boldsymbol{R}_\phi$ about $\Delta\phi$ around $\hat{\boldsymbol{e}}_2$ and a rotation $\boldsymbol{R}_\theta$ about $\Delta\theta$ around $\hat{\boldsymbol{\omega}}_\theta = \boldsymbol{R}_\phi \hat{\boldsymbol{e}}_1$, where $\hat{\boldsymbol{e}}_k$ denotes Euclidean unit vectors.

In addition to the exterior orientation $\boldsymbol{R}_c$, $\boldsymbol{t}_c$, the camera intrinsics f_u, f_v, c_u, c_v (recall Section 2.2.1) are required to describe the perspective projection. The principal point

$$c_u = \frac{w_e - 1}{2}, \tag{5.10}$$

$$c_v = \frac{h_e - 1}{2} \tag{5.11}$$

in the center of an image $\boldsymbol{I}_e$ is defined by its width w_e and height h_e. For all results in this thesis; the image dimensions are $w_e = 256$ and $h_e = 192$. Further, a diagonal FOV of 85° is used, approximating the focal length of non-wide-angle cameras in modern smartphones. The FOV and image dimensions define the focal length in pixels (recall Equation (2.15)). In turn, the camera intrinsics define the camera calibration matrix (recall Equation (2.17)). As described in Section 2.2.1, a homogeneous world coordinate point $\tilde{\boldsymbol{x}}_w \in \mathbb{P}^3$ can be projected into a homogeneous pixel coordinate $\tilde{\boldsymbol{u}} \in \mathbb{P}^2$ using the camera matrix $\boldsymbol{P} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R}_c & \boldsymbol{t}_c \end{bmatrix}$ (recall Equation (2.19)). The inverse transformation

$$\tilde{\boldsymbol{d}}_w = \boldsymbol{P}^{-1} \tilde{\boldsymbol{u}} \tag{5.12}$$

yields a homogeneous 3D direction vector $\tilde{\boldsymbol{d}}_w$ with undefined spatial depth ($\tilde{\boldsymbol{d}}_w = (\boldsymbol{d}_w, 0)^\mathsf{T}$) for a homogeneous pixel coordinate $(u, v, 1)^\mathsf{T}$. Thus, for each pixel in image $\boldsymbol{I}_e$, a homogeneous direction vector $\tilde{\boldsymbol{d}}_w$ in world coordinates can be determined. The coordinates at which the direction vector $\tilde{\boldsymbol{d}}_w$ intersects the unit sphere corresponds to a pixel $(u_p, v_p)^\mathsf{T}$ in image $\boldsymbol{I}_p$ (see Equations (5.5) and (5.6)). This allows mapping each pixel in image $\boldsymbol{I}_e$ to a pixel in panorama $\boldsymbol{I}_p$. Figure 5.5 illustrates this process. The red frame in 5.5a corresponds to the red frame in 5.5b. The resulting image $\boldsymbol{I}_e$ serves as an input image for the dataset.

Multiple input images can be cropped from one panorama to increase the amount of potential training data. The following section will demonstrate why different input images from the same panorama result in different lighting parameters, even though they are directly derived from the panorama. As in [GSY+17], eight different image excerpts are taken from each panorama as input images. Therefore, eight different view directions (θ_0, ϕ_0) are randomly chosen, as normally distributed random variables. The polar angle θ_0 ranges from $\frac{\pi}{3}$ to $\frac{2\pi}{3}$, and the azimuthal angle ϕ_0 ranges from $-\frac{\pi}{2}$ to $\frac{3\pi}{2}$. If the deviation between two viewing angle pairs i and j is:

$$\sqrt{(\theta_{0,i} - \theta_{0,j})^2 + (\phi_{0,i} - \phi_{0,j})^2} < 10^\circ, \tag{5.13}$$

a new pair is chosen to ensure sufficient variation in the view directions. From the original 2100 panoramas in the Laval Indoor HDR Dataset, this process yields 16 800 input images for the new dataset.

5.4.3 | Lighting Parameters

The corresponding lighting parameters must be determined for each image excerpt. 360° HDR panoramas are suitable for describing the lighting situation of the scene from the point where the camera tripod is located, the panorama center. However, this does not correspond to the scene around the image excerpt. A virtual object placed in the image excerpt is located at a virtual insertion point inside the excerpt and not at the center of the panorama. Determining the lighting situation at the virtual insertion point requires a light probe at that location, as discussed in Section 5.2. Alternatively, a new panorama must be captured with the tripod positioned at that point. For the panoramas in the dataset, it is impossible to capture new images without access to the physical location, so the lighting situation must be approximated differently.

One way to approximate the lighting situation at a new insertion point is to create a new panorama around this point by projecting from the original panorama. In [GSY+17], a warping operator is defined to describe this projection. As with many approximations, some assumptions must be made that limit the complexity of the problem. First, all physical points described by pixels in the original panorama are assumed to be equidistant from the panorama center. Secondly, the effects of occlusions from existing objects play a subordinate role in the given scenes. These assumptions are suitable for the sought light sources, which are usually located on the ceilings or walls of the rooms.

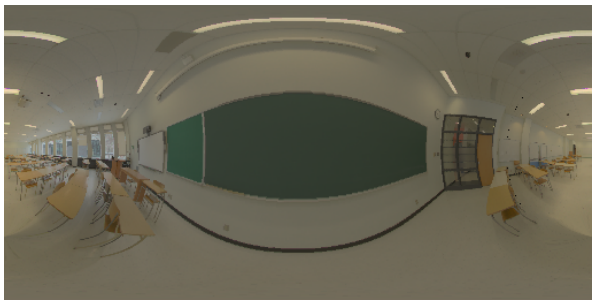
In the following, the original HDR panorama is again referred to as image $\boldsymbol{I}_p$. The warped HDR panorama around the new insertion point is referred to as image $\boldsymbol{I}_w$ with dimensions $w_w \times h_w$. For the results in this thesis, a width of $w_w = 1024$ and a height of $h_w = 512$ is used. Considering the 3D world coordinate space of $\boldsymbol{I}_p$, with the original camera center placed at the origin $\mathbf{0}$, all points of the panorama lie on the unit sphere

$$x_p^2 + y_p^2 + z_p^2 = 1\,, \tag{5.14}$$

when assuming an equidistant scene. The warped panorama should have its center at $\boldsymbol{x}_c = (x_c, y_c, z_c)^\mathsf{T}$ inside the unit sphere $|\boldsymbol{x}_c| \leq 1$, corresponding to the position of the insertion point. This point represents the location of a virtual camera that would have captured the warped panorama. Each pixel of the warped panorama $\boldsymbol{I}_w$ corresponds to a view direction $\boldsymbol{d}_w$ from the warped panorama center $\boldsymbol{x}_c$. For such a direction $\boldsymbol{d}_w$, the

(a) Original panorama $\boldsymbol{I}_p$

(b) Image excerpt $\boldsymbol{I}_e$



(c) Warped panorama $\boldsymbol{I}_{w'}$

Figure 5.6: A rectilinear image excerpt $\boldsymbol{I}_e$ (b) is extracted from a given equirectangular panorama $\boldsymbol{I}_p$ (a). Around a virtual insertion in the center of $\boldsymbol{I}_e$, a warped panorama $\boldsymbol{I}_{w'}$ (c) is projected and rotated in such a way that the view direction of $\boldsymbol{I}_e$ is in the center of $\boldsymbol{I}_{w'}$. [Original panorama sourced from the Laval HDR Indoor Dataset.]

intersection of the ray

$$\boldsymbol{r}_{cw}(\tau) = \boldsymbol{x}_c + \tau \boldsymbol{d}_w \tag{5.15}$$

with the unit sphere at the origin can be determined:

$$\left((\tau d_{w_x} + x_c)^2 + (\tau d_{w_y} + y_c)^2 + (\tau d_{w_z} + z_c)^2\right) = 1\,. \tag{5.16}$$

The resulting quadratic equation

$$\tau^2 + \frac{2(d_{w_x} + d_{w_y} + d_{w_z})}{d_{w_x}^2 + d_{w_y}^2 + d_{w_z}^2}\tau + \frac{x_c^2 + y_c^2 + z_c^2 - 1}{d_{w_x}^2 + d_{w_y}^2 + d_{w_z}^2} = 0 \tag{5.17}$$

for the unknown τ has two solutions, corresponding to the two intersection points of the ray with the unit sphere. The positive solution corresponds to the point in the virtual

camera's line of sight, and the negative solution corresponds to the intersection point behind the virtual camera. By substituting the positive solution τ_+ into the ray equation (5.15), a point $\boldsymbol{x}_p = \boldsymbol{r}_{cw}(\tau_+)$ on the unit sphere for a view direction $\boldsymbol{d}_w$ can be determined. Each point on the unit sphere corresponds to a pixel in image $\boldsymbol{I}_p$ (see Equations (5.5) and (5.6)). Since a ray intersection with the unit sphere can be determined for every pixel in the warped panorama $\boldsymbol{I}_w$, each pixel in $\boldsymbol{I}_w$ maps to a corresponding pixel in image $\boldsymbol{I}_p$ (see Figure 5.6).

If the lighting parameters are extracted from this panorama $\boldsymbol{I}_w$, the resulting light direction is relative to the absolute world coordinate system. Since the network receives only an input image $\boldsymbol{I}_e$ without reference to the world coordinate system, the light direction relative to the local camera coordinate system is preferred. One way to determine the parameters relative to the camera coordinate system of image $\boldsymbol{I}_e$ is to rotate the warped panorama $\boldsymbol{I}_w$ so that its image center aligns with the view direction (θ_0, ϕ_0) of image $\boldsymbol{I}_e$. It is important to note that due to the projection of the spherical texture onto the flat surface of the 2D image in an equirectangular panorama such as image $\boldsymbol{I}_w$, linearly shifting the pixels is not feasible. In the following, the rotated warped panorama is denoted as image $\boldsymbol{I}_{w'}$. To rotate image $\boldsymbol{I}_w$ by an angle pair $(\Delta\theta, \Delta\phi)$, it is necessary to map each pixel $(u_w, v_w)^\mathsf{T}$ of $\boldsymbol{I}_w$ onto the unit sphere (see Equations (5.5) and (5.6)). This location corresponds to Cartesian coordinates (recall Equations (2.4) - (2.6)), which can then be appropriately rotated:

$$\boldsymbol{x}_{w'} = \boldsymbol{R}_\phi \boldsymbol{R}_\theta \, \boldsymbol{x}_w \, . \tag{5.18}$$

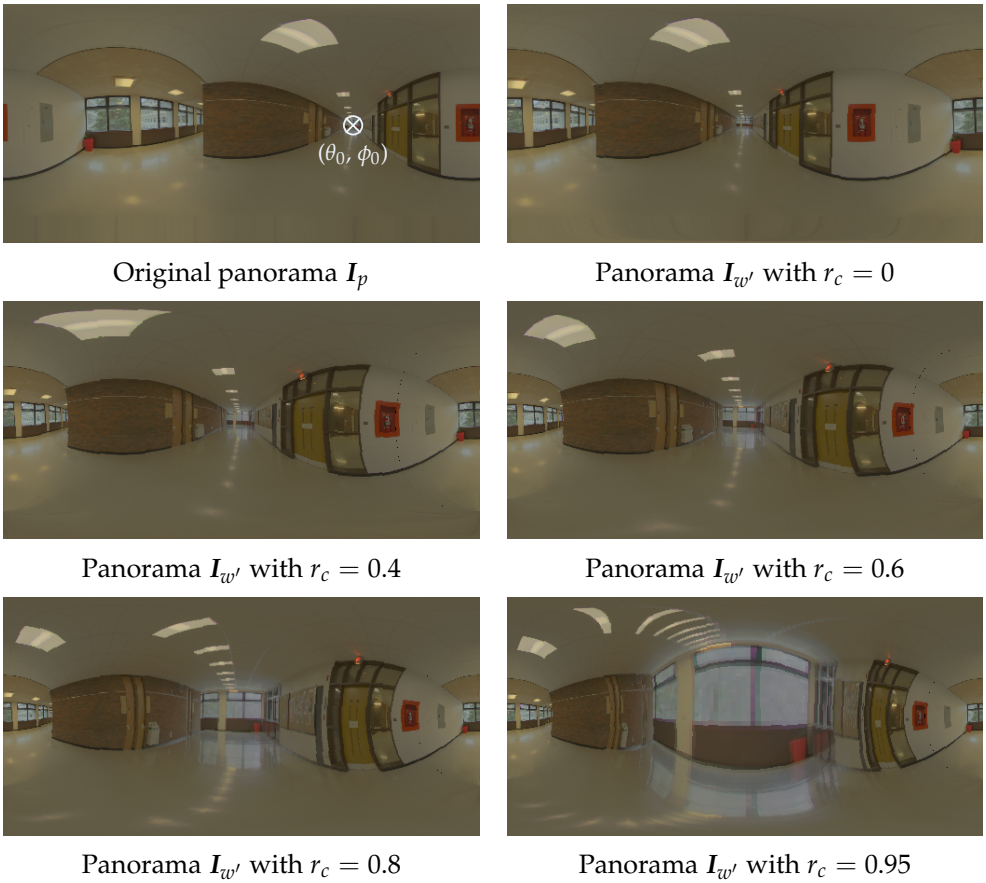
Here, the rotation matrices $\boldsymbol{R}_\phi$ and $\boldsymbol{R}_\theta$ correspond to the matrices already used in Equation (5.9). As rotations are length-preserving affine transformations, the rotated point $\boldsymbol{x}_{w'}$ also lies on the unit sphere. Subsequently, the new point can be transformed back into unit spherical coordinates with Equations (2.8) and (2.9). This location on the unit sphere corresponds to a pixel

$$u_{w'} = \left\lfloor \left(\frac{\phi_{w'} + \pi/2}{2\pi} \right) w_{w'} \right\rceil \, , \tag{5.19}$$

$$v_{w'} = \left\lfloor \left(\frac{\theta_{w'}}{\pi} \right) h_{w'} \right\rceil \tag{5.20}$$

in panorama $\boldsymbol{I}_{w'}$ with dimensions $w_{w'} \times h_{w'}$. Therefore, each pixel $(u_{w'}, v_{w'})^\mathsf{T}$ in the rotated panorama $\boldsymbol{I}_{w'}$ maps to a pixel $(u_w, v_w)^\mathsf{T}$ in panorama $\boldsymbol{I}_w$. Figure 5.6 shows a triple comprising matching images $\boldsymbol{I}_p$, $\boldsymbol{I}_e$, and $\boldsymbol{I}_{w'}$.

It is necessary to know the point $\boldsymbol{x}_c$ at which an object would be placed for a given image excerpt $\boldsymbol{I}_e$ to create the warped panorama $\boldsymbol{I}_{w'}$. However, there is not only a single



Original panorama $\boldsymbol{I}_p$ | Panorama $\boldsymbol{I}_{w'}$ with $r_c = 0$

Panorama $\boldsymbol{I}_{w'}$ with $r_c = 0.4$ | Panorama $\boldsymbol{I}_{w'}$ with $r_c = 0.6$

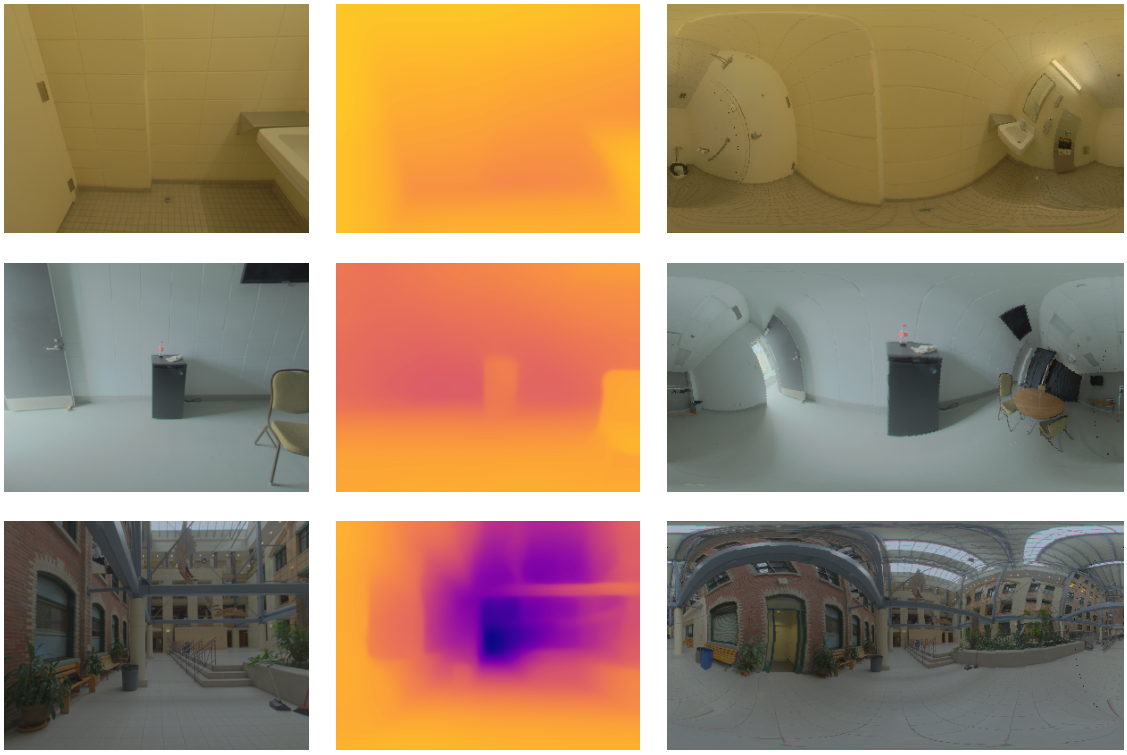
Panorama $\boldsymbol{I}_{w'}$ with $r_c = 0.8$ | Panorama $\boldsymbol{I}_{w'}$ with $r_c = 0.95$

Figure 5.7: For a given panorama $\boldsymbol{I}_p$ with marked view direction (θ_0, ϕ_0), warped panoramas $\boldsymbol{I}_{w'}$ with varying insertion points, depending on the parameter r_c, are created. [Original panorama sourced from the Laval HDR Indoor Dataset.]

possible insertion point for a given excerpt. It is a user or application-dependent choice. Therefore, an additional assumption is made that the virtual object is placed in the center of image $\boldsymbol{I}_e$ when the lighting estimation is performed. The insertion point

$$\boldsymbol{x}_c = \begin{pmatrix} r_c \sin\theta_0 \cos\phi_0 \\ r_c \cos\theta_0 \\ r_c \sin\theta_0 \sin\phi_0 \end{pmatrix} \tag{5.21}$$

can thus be described depending on the view direction (θ_0, ϕ_0). The parameter r_c considers the unknown spatial depth of the panorama. A value of $r_c = 1$ corresponds to a virtual camera position at the extremum of the panorama. This corresponds, for example, to placing the virtual camera directly against a wall, resulting in significant distortions.



(a) Image excerpt I_e (b) Depth map (c) Warped panorama $I_{w'}$

Figure 5.8: For panoramas of varying room sizes, an image excerpt I_e (a) is selected, and a depth map (b) is generated from it. The color coding of the depth map follows the color scheme ranging from light yellow for 0 m to dark violet for 10 m. A warped panorama (c) is created based on the depth information. [Original panoramas sourced from the Laval HDR Indoor Dataset.]

A parameter of $r_c = 0$ corresponds to a new camera position at the origin of the original panorama's camera and, therefore, no warping. Figure 5.7 illustrates different values of r_c for the warped panorama $I_{w'}$. For a straightforward baseline solution, the assumption can be made that a virtual object is consistently placed at fixed distance from the camera, and all rooms are approximately the same size. Consequently, r_c can be set as a constant across all images, for example, by setting $r_c = 0.8$.

A more precise understanding of the scene geometry is needed to identify an image-specific insertion point x_c to obtain more realistic lighting parameters. One approach involves using a depth map of image I_e. Given the absence of actual depth information for the panoramas within the dataset, it is necessary to estimate depth values. As of the

time of this writing, DistDepth[3] represents the state-of-the-art method for approximating depth maps from monocular perspective images, particularly for indoor scenes. Here, a perspective image (e.g., image $\boldsymbol{I}_e$) is fed into a neural network, producing a well-defined depth map with values ranging from 0 m to 10 m. For a detailed insight into the structure and functionality of DistDepth, refer to [WWH+22]. Figure 5.8 shows for a given images $\boldsymbol{I}_e$ in 5.8a the estimated depth maps in 5.8b. The *plasma* color scale from the Virdis color palette [SW15; NAR18] is used to visualize the scalar depth values. The color scale uses light yellow for the smallest values (i.e., 0 m) and dark violet for the largest values (i.e., 10 m).

Since the camera center for the panoramas in the dataset is typically positioned near the center of the room, the depth map of the image excerpt $\boldsymbol{I}_e$ provides insights into the room's size. The maximum depth of the depth map is referred to as $d_{\max}$ in the following. In the case of a considerably large room, this depth falls within the range of the maximum possible value (10 m). In such a scenario, when a virtual object is placed in front of the camera, the lighting conditions at that point are relatively similar to the lighting conditions at the origin of the panorama (i.e., $r_c \approx 0$). Conversely, in a small room (e.g., $d_{\max} = 1\,\text{m}$) the lighting conditions at the insertion point in front of the camera change significantly, necessitating a more pronounced warping (i.e., $r_c \approx 1$). From these two conditions, different relationships between r_c and $d_{\max}$ can be modeled. Among others, quadratic, inverse quadratic, and sinusoidal models were tested. However, a linear relationship

$$r_c = -\frac{1}{10} d_{\max} + 1 \tag{5.22}$$

gave comparable results and was used for its simplicity and interpretability. Figure 5.8 shows the corresponding depth maps and warped panoramas $\boldsymbol{I}_{w'}$ based on the depth information for images $\boldsymbol{I}_e$, illustrating scenarios for small, medium, and large rooms.

The warped HDR panorama $\boldsymbol{I}_{w'}$ determines the lighting parameters for an input image $\boldsymbol{I}_e$ (recall Section 5.3). Initially, the overall brightness of individual pixels with respect to human perception—the luminance $L_{\text{HDR}}(u, v)$—is determined (recall Equation (5.2)). From the resulting luminance map L_{HDR}, areas with less than 5 % of the maximum luminance in the map are masked out. The remaining highlight area is used to determine the average light direction. Therefore, each pixel is again assigned a direction as a Cartesian unit vector. Section 5.5 explains why directions are specified as Cartesian vectors rather than spherical coordinates. The directions of the pixels in the highlight area must be weighted to determine the average light direction. Initially, each direction

[3]The code is available at: `https://github.com/facebookresearch/DistDepth`

should be weighted according to its intensity $L_{\text{HDR}}(u, v)$. However, a single weight is insufficient, as the equirectangular projection of the map causes not every pixel to occupy an equally sized area on the unit sphere. A region near the poles occupies significantly more pixels than one near the equator. This is compensated by a second weighting

$$\omega_{\text{area}}(v) = \frac{2\pi^2}{w_{w'} h_{w'}} \sin\left(\frac{v + 0.5}{h_{w'}} \pi\right) \tag{5.23}$$

considering the area that a pixel occupies on the unit sphere. The weighted average light direction corresponds to parameter $\hat{\boldsymbol{d}}_l$ for the input image $\boldsymbol{I}_e$ with the warped panorama $\boldsymbol{I}_{w'}$.

The warped HDR panorama is converted into a tone-mapped LDR panorama (recall Section 5.4.2) to determine the light color $\boldsymbol{c}_l$. Subsequently, the identical regions as before are masked out from the LDR image, leaving only the highlight area. The exact weights used before can be applied to the individual RGB channels of the LDR image to obtain the average light color $\boldsymbol{c}_l$.

The ambient light color $\boldsymbol{a}_l$ describes the average color effect of the scene, excluding the main light source. The LDR panorama is also used to determine it. In this case, the masking is inverted to mask the highlight area. The average RGB value $\boldsymbol{a}_l$ of the remaining pixels can be calculated using the same weighting.

The contrast level o_l characterizes the ratio between the intensity of light sources and the remaining areas of the scene. The luminance $L^{(v)}_{\text{HDR}}(u, v)$ describes the intensity of individual pixels as a scalar value. To obtain a measure of the ratio between highlight area and the rest of the scene, the total luminance $L^{(v)}_{\text{tot}}$ of each area is first determined. The individual luminance values of the highlight area are added up to $L^{(v)}_{\text{tot},h}$ as a weighted sum, again with weights $\omega_{\text{area}}(v)$ for the area that a pixel occupies on the unit sphere. Similarly, the total luminance of the ambient area $L^{(v)}_{\text{tot},a}$ is determined. The contrast level

$$o_l = 1 - \tanh\left(\frac{L^{(v)}_{\text{tot},a}}{0.05 \cdot L^{(v)}_{\text{tot},h}}\right) \tag{5.24}$$

includes the quotient of the two weighted sums. The scalar factor of 0.05 was determined empirically so that bright areas do not dominate the contrast level. The tanh function is applied to limit the range of the quotient between 0 and 1. The less the luminance of the highlight area differs from the luminance of the ambient area, the lower the contrast level.

5.5 | Neural Network

The previous section described a dataset comprising pairs of input images and associated lighting parameters. This dataset is used to train a neural network to predict the lighting parameters for a given input image. Therefore, it is necessary to use an appropriate network architecture.

5.5.1 | Network Architecture

Several factors influence the choice of an appropriate network architecture. In addition to the nature and complexity of the problem, the characteristics of the dataset and the computational resources available on the target platform play a crucial role. Therefore, the requirements for the network architecture for lighting estimation should first be discussed briefly.

The input data for the network consists of image data. For image processing tasks, networks with convolutional layers are commonly used (recall Section 2.4.2). The output of the network should correspond to the parameters of the lighting representation. In image classification networks, multiple scalar outputs also exist for each input image. Typically, each neuron in the output layer corresponds to a different class. The tasks of image classification and lighting estimation are thus closely related in their inputs and outputs, making similar architectures suitable.

The dataset comprises 16 800 tuples of input images and lighting parameters minus a percentage for validation and test data. This amount is insufficient to address the complexity of deeper image classification networks, which are often trained on datasets that include several orders of magnitude more data, such as ImageNet [DDS+09] with about 14 million input images. However, when using a network with complexity adapted to the dataset size, the number of feature maps is so low that insufficient individual features can be identified to characterize the lighting situation. In computer vision, this issue is frequently overcome by using pre-trained networks. Instead of training the network from scratch with the specific dataset, the network is initially trained with a larger dataset (e.g., ImageNet). Networks pre-trained on diverse datasets have learned robust features that generalize well for various problems. Subsequently, the network is fine-tuned with the actual dataset to address the specific challenges of the new task.

An additional requirement for the network is interference on AR devices since the lighting estimation is intended for AR applications. The network should only consume a fraction

of the available resources, as a larger portion is needed for other tasks (e.g., rendering virtual objects). The network's interference time should be in interactive frame rates since the lighting situation may need to be reassessed frequently, for example, when the position of the virtual object and the orientation of the camera change significantly.

Currently, the most significant state-of-the-art network architectures for image classification tasks that use convolutional layers include AlexNet [KSH12], VGGNet [SZ15], GoogLeNet [SLJ+15], ResNet [HZR+16], and DenseNet [HLW17]. The DenseNet architecture was used, among other reasons, for its parameter efficiency and fast inference speed. AlexNet and VGGNet have significantly higher parameter numbers, leading to increased memory and computational demands. While GoogLeNet and other Inception-based architectures are highly efficient, their complex structure makes them more challenging to modify and optimize for resource-constrained devices. ResNet, like DenseNet, uses skip connections between non-adjacent layers. However, ResNet's residual connections additively merge the outputs of previous layers, whereas DenseNet concatenates feature maps from previous layers (recall Section 2.4.3). This allows DenseNet to access previously extracted features at any point in the network, whereas ResNet may generate redundant feature maps. By reusing features, DenseNet effectively reduces the total number of parameters, which helps reduce overfitting, particularly on small datasets. Additionally, DenseNet achieves competitive accuracy at lower computational cost, making it well-suited for deployment on constrained resources of AR devices.

As mentioned above, for a relatively small dataset like the one described in Section 5.4, it is advisable to pre-train the feature maps on a larger and more diverse dataset. ImageNet, as a vast dataset with a variety of input images from a wide range of categories, is a good choice for acquiring generic features. Section 2.4.3 provides a detailed description of the DenseNet architecture for training on ImageNet.

The network architecture needs to be slightly modified to fine-tune the network with the lighting estimation dataset. The classifier is removed and its weights and biases are discarded since only the pre-trained feature maps are of interest. Distinct output neurons for each parameter are needed to predict the lighting parameters described in Section 5.3 with the network. Therefore, the result of the feature extracted is passed to a latent vector analogous to one in the classifier. Subsequently, the latent vector is forwarded through a fully connected layer with a size of 512. Batch normalization is applied to the fully connected layer, and its output is activated using an ELU activation (recall Equation (2.27)). Each parameter—$\hat{\boldsymbol{d}}_l$, $\boldsymbol{c}_l$, $\boldsymbol{a}_l$, and o_l—is assigned its distinct output, referred to as a *network head*. In each network head, all neurons are fully connected to the preceding layer,

but no connections exist between individual heads. The head for a vectorial parameter comprises three neurons and for the scalar parameter o_l one neuron. The individual components of the color parameters $\boldsymbol{c}_l$ and $\boldsymbol{a}_l$, as well as the contrast level o_l, are scaled to a value range between 0 and 1 through sigmoid activation. The individual components of the Cartesian direction vector $\hat{\boldsymbol{d}}_l$ are scaled into a range between -1 and 1 through tanh activation. Subsequently, the entire vector is normalized.

5.5.2 | Training Procedure

The network training is divided into two stages. In the first stage—*pre-training*—the network is trained for image classification. Therefore, a classifier is attached to the feature extractor. In this case, the architecture corresponds to the DenseNet-BC-121 described in [HLW17]. ImageNet [DDS+09], comprising 14 million input images distributed across 1000 distinct classes, is the dataset in this stage. During the network training, a loss function is minimized. In Section 2.4.1, the MSE loss was given as an example. The MSE is suitable for minimizing the distance between predicted and ground truth values. For image classification, the prediction represents a probability distribution across multiple classes that needs to be compared with the ground truth probability distribution. For probability distributions, the MSE loss is less suitable; instead, the cross-entropy loss

$$\mathcal{L}_{\mathrm{CE}} = -\log\left(\psi_{i=\mathrm{gt}}\right) \tag{5.25}$$

is commonly used for classification tasks. Here, ψ_i represents the softmax-normalized output of the i-th class (recall Equation (2.34)), where i corresponds to the correct ground truth class for the input image. The cross-entropy loss penalizes the network more strongly when a low probability is assigned to the correct class. For a detailed analysis of cross-entropy loss in classification networks, refer to [JC16].

In this stage, the network trains for 100 epochs, using a mini-batch size of 128. The initial learning rate $\lambda_{\mathrm{lr}} = 0.1$ is divided by 10 at epochs 30, 60, and 90. More details on the training of the DenseNets for image classification can be found in [HLW17]. It is noteworthy that, at the time of writing, pre-trained DenseNet weights for ImageNet are available in standard machine learning libraries such as PyTorch[4], eliminating the need for individual pre-training.

In the second stage—*fine-tuning*—the network is trained for lighting estimation. Therefore, the classifier of the network is removed and replaced with the structure described

[4]PyTorch is available at: `https://pytorch.org/`

in Section 5.5.1, incorporating multiple network heads. During fine-tuning, the dataset described in Section 5.4 is used as training data. Initially, 5 % of the dataset is set aside for later evaluation as test data. The remaining data is randomly split into training data and validation data using an 85 %/15 % split. During training, the estimated parameters—$\hat{\boldsymbol{d}}_l^{(\text{est})}$, $\boldsymbol{c}_l^{(\text{est})}$, $\boldsymbol{a}_l^{(\text{est})}$, and $o_l^{(\text{est})}$—and the ground truth parameters—$\hat{\boldsymbol{d}}_l^{(\text{gt})}$, $\boldsymbol{c}_l^{(\text{gt})}$, $\boldsymbol{a}_l^{(\text{gt})}$, and $o_l^{(\text{gt})}$—are compared directly. The MSE loss is suitable since distances between these values are to be minimized. The overall loss is the weighted sum of individual parameter losses:

$$\begin{aligned}\mathcal{L} \;&= w_d \mathcal{L}_{\text{MSE}}(\hat{\boldsymbol{d}}_l^{(\text{est})}, \hat{\boldsymbol{d}}_l^{(\text{gt})}) + w_c \mathcal{L}_{\text{MSE}}(\boldsymbol{c}_l^{(\text{est})}, \boldsymbol{c}_l^{(\text{gt})}) \\ &\quad + w_a \mathcal{L}_{\text{MSE}}(\boldsymbol{a}_l^{(\text{est})}, \boldsymbol{a}_l^{(\text{gt})}) + w_o \mathcal{L}_{\text{MSE}}(o_l^{(\text{est})}, o_l^{(\text{gt})})\,.\end{aligned} \tag{5.26}$$

Different parameters have varying importance and are, therefore, weighted differently, denoted by weights $w_d = 5$, $w_c = 2$, $w_a = 2$, and $w_o = 1$. The most important parameter is the light direction, as it plays a crucial role in achieving proper illumination and is essential for casting shadows (see Section 6.3). The light and ambient color are equally significant, contributing to the overall scene mood. The contrast level is of secondary importance, as parameter fluctuations are perceived less strongly.

At this point, the reason for defining the parameter for the direction of light $\hat{\boldsymbol{d}}_l$ in Cartesian coordinates instead of unit spherical coordinates in Section 5.3 becomes evident. Spherical coordinates have a discontinuity in the azimuthal angle ϕ at the transition from 2π to 0, which can lead to incorrect distances between two points without modification of the MSE loss. Even with appropriate modification of the MSE, training a network with the light direction in spherical coordinates can lead to unstable predictions for the direction, as described in [GHS+19].

In the second stage, the network is further trained for an additional 60 epochs, also with a batch size of 128. The initial learning rate $\lambda_{\text{lr}} = 0.001$ is halved every 15 epochs. Since the dataset used for fine-tuning is considerably smaller than the ImageNet in the first stage, training these 60 epochs is significantly faster than training in the first stage. The fine-tuning process takes two hours on two Nvidia RTX A6000 GPUs, while pre-training takes several days.

5.6 | Evaluation

A comprehensive evaluation is necessary to assess the quality of the lighting estimation method. This evaluation is based on the 5 % of the dataset initially set aside as test

data, which remained unseen during training. Initially, ground truth and prediction are qualitatively compared. Therefore, the light direction $\hat{\boldsymbol{d}}_l$, light color $\boldsymbol{c}_l$, and ambient color $\boldsymbol{a}_l$ are visualized. Projecting the light source onto an environment map gives a 2D representation of the light source in 3D space. One possible approach involves modeling the radiance distribution of the light source as a spherical Gaussian [WRG+09; GHS+19]

$$G_\Omega = \alpha_\Omega \exp \lambda_\Omega \left(\boldsymbol{\mu}_\Omega^\intercal \boldsymbol{x} - 1 \right) , \tag{5.27}$$

where α_Ω is the amplitude corresponding to the intensity of individual color channels of the light color. The spread of the distribution $\lambda_\Omega = 2\pi$ is chosen for visualization purposes. The lobe axis $\boldsymbol{\mu}_\Omega$ corresponds to the light direction. Thus, a color value can be determined for every point $\boldsymbol{x}$ on the unit sphere. If $\boldsymbol{x}$ aligns with the light direction $\hat{\boldsymbol{d}}_l$, the scalar product $\hat{\boldsymbol{d}}_l^\intercal \boldsymbol{x}$ maximizes the color intensity. For deviating directions, the intensity decreases exponentially.

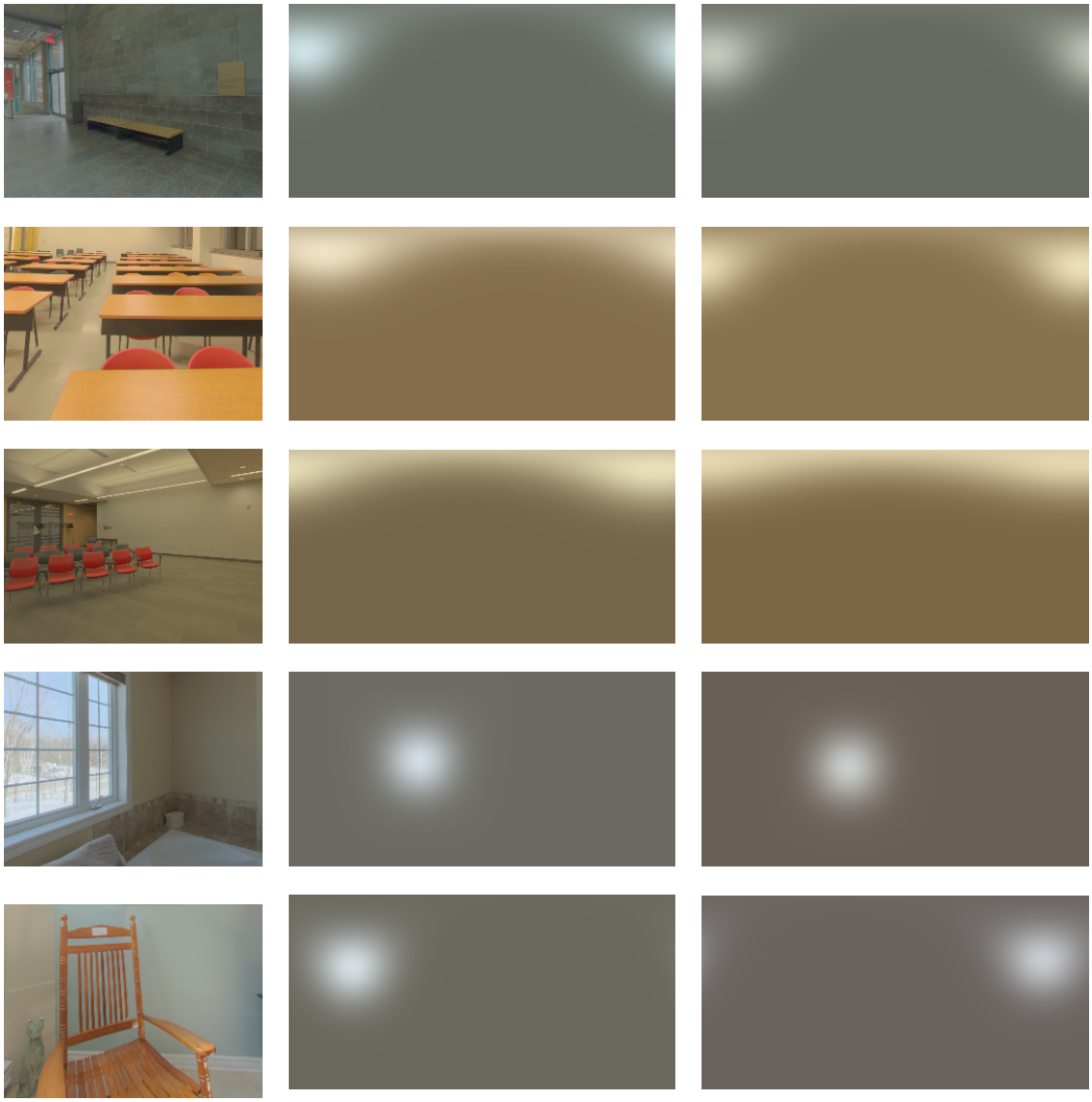
Let $\boldsymbol{I}_{g'}$ denote the environment map with the resolution $w_g \times h_g$; each pixel (u, v) can be assigned a Cartesian coordinate on the unit sphere similarly to before using Equations (5.5), (5.6), and (2.4) - (2.6). Exploiting the relations $\cos(x - \pi/2) = \sin x$ and $\sin(x - \pi/2) = -\sin x$ leads to:

$$\boldsymbol{I}_{g'}(u, v) = \boldsymbol{c}_l \exp 2\pi \left(\left(\sin \frac{\pi v}{h_g} \sin \frac{2\pi u}{w_g}, \cos \frac{\pi v}{h_g}, -\sin \frac{\pi v}{h_g} \cos \frac{2\pi u}{w_g} \right) \hat{\boldsymbol{d}}_l - 1 \right) . \tag{5.28}$$

To additionally represent the ambient color in the environment map, the ambient color must be superimposed onto $\boldsymbol{I}_{g'}$. In image editing, different blending modes are used to overlay two images. The *screen* blending mode is particularly suitable for blending dark areas such as the constant ambient color $\boldsymbol{a}_l$ with highlights such as in $\boldsymbol{I}_{g'}$ without causing them to exceed the color range. Screen blending results in an environment map

$$\boldsymbol{I}_g(u, v) = \boldsymbol{c}_w - (\boldsymbol{c}_w - \boldsymbol{I}_{g'}(u, v)) \odot (\boldsymbol{c}_w - \boldsymbol{a}_l) , \tag{5.29}$$

where $\boldsymbol{c}_w$ denotes the white point $(1, 1, 1)^\intercal$. For more information on blending modes, refer to [W24].



(a) Input image (b) GT map $I_g^{(gt)}$ (c) Estimated map $I_g^{(est)}$

Figure 5.9: For a given input image (a), the ground truth lighting parameters from the dataset are visualized in an environment map $I_g^{(gt)}$ (b), and in comparison, the lighting parameters estimated by the network are visualized in map $I_g^{(est)}$ (c). The rows are arranged in descending order of estimation accuracy.

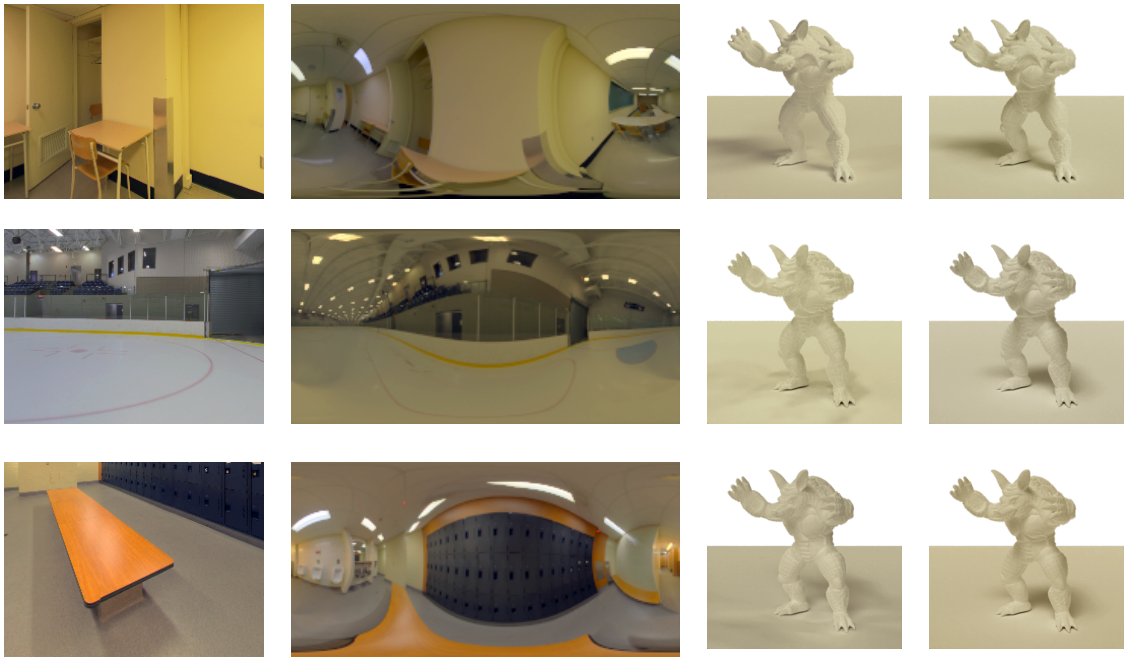
Figure 5.9 shows the environment map visualization in (b) for the ground truth lighting parameters of five input images in (a). For comparison, the lighting parameters estimated by the network are visualized in Figure 5.9c. The order of the rows is arranged from top to bottom based on descending prediction accuracy, measured by the loss function (see Equation (5.26)). For the first row, an image from the top 10 % estimation accuracy of the test data was selected, and for the last row, an image from the worst 10 %. The three images in the middle were selected from a range of average accuracy.

The qualitative evaluation aims to understand of how well the method generalizes concerning the lighting parameters described in Section 5.3. The subsequent quantitative evaluation assesses how suitable this simplified representation is for approximating the complex lighting situations of indoor scenes. For this purpose, a simple experimental setup is proposed: a 3D object is placed on a flat plane within an entirely virtual 3D scene.

Each input image in the test set has a corresponding warped panorama (recall Section 5.4), which represents the actual lighting conditions at the insertion point within the image. Rendering the experimental setup with this panorama as an environment map using path tracing with environment sampling (recall Section 2.3.3) results in a ground truth rendering of the virtual object, referred to as image $\boldsymbol{I}_r^{(\mathrm{gt})}$. The estimated lighting parameters, on the other hand, are obtained by feeding the input image into the network. The experimental setup can also be rendered with a light source initialized from these lighting parameters. This involves placing a soft directional light (see Section 6.4) in the estimated light direction $\hat{\boldsymbol{d}}_l^{(\mathrm{est})}$ with the estimated light color $\boldsymbol{c}_l^{(\mathrm{est})}$, along with setting a constant ambient color based on parameter $\boldsymbol{a}_l^{(\mathrm{est})}$. The resulting rendering of the estimated lighting is referred to as image $\boldsymbol{I}_r^{(\mathrm{est})}$. Both images have the same resolution $w_r \times h_r$. Figure 5.10 shows three examples of input images in (a), their corresponding warped panoramas in (b), ground truth renderings $\boldsymbol{I}_r^{(\mathrm{gt})}$ in (c), and renderings with estimated lighting parameters $\boldsymbol{I}_r^{(\mathrm{est})}$ (d). The Stanford Armadillo[5] [KL96] is the 3D object for the setup.

The image pairs could also be qualitatively evaluated. While qualitative image evaluations are well-suited for exploring aesthetic aspects that are difficult to quantify, they inherently involve a personal bias based on the viewer's preferences. Quantitative metrics are more appropriate for obtaining an unbiased and objective comparison between two sets of images. These metrics can be calculated automatically and produce repro-

[5]The 3D object is publicly available at: `https://graphics.stanford.edu/data/3Dscanrep/`



(a) Input image (b) GT environment map (c) $I_r^{(gt)}$ (d) $I_r^{(est)}$

Figure 5.10: For a given input image (a), a 3D scene with an armadillo on a plane is rendered in (c) with the corresponding environment map (b) using IBL techniques and in (d) with a parametric light source estimated from the input image. [Images sourced from [SSS23].]

ducible and comparable results. For additional information on image comparison, refer to [WBS+04].

In the following, four widely used metrics are employed, each with its own strengths and weaknesses. The root-mean-square error (RMSE)

$$\epsilon_{\text{RMSE}} = \sqrt{\frac{1}{w_r \cdot h_r} \sum_u \sum_v \left(\boldsymbol{I}_r^{(gt)}(u,v) - \boldsymbol{I}_r^{(est)}(u,v) \right)^2} \tag{5.30}$$

describes the deviation of intensity values. The lower this error value, the smaller the difference between the two images, indicating a better reproduction of the exact lighting situation by the estimated parametric lighting representation. The standard RMSE penalizes larger errors more than smaller ones, providing a good measure to assess the relationship between ambient and light intensity. The scale-invariant root-mean-square error (si-RMSE)

$$\epsilon_{\text{si-RMSE}} = \frac{\text{RMSE}}{\sqrt{\sum_u \sum_v \left(\boldsymbol{I}_r^{(gt)}(u,v) \right)^2}} \tag{5.31}$$

Method	$\epsilon_{\textbf{RMSE}}$	$\epsilon_{\textbf{si-RMSE}}$	$\epsilon_{\textbf{RMLE}}$	$\epsilon_{\textbf{RGB}^\circ}$
[SSS23]	0.1101	0.1501	0.069 28	3.542°
[GHS+19](1)	0.1114	0.1518	0.070 07	3.556°

Table 5.1: Comparison by four different metrics of renderings of an experimental setup using the proposed lighting estimation ([SSS23]) versus the state-of-the-art parametric lighting estimation ([GHS+19]) for a single light source.

evaluates pixel-wise dissimilarities between the two images independent of the intensity scale by scaling them by the root-mean-square of the ground truth image. Consequently, different exposure levels in the two images, for example, due to different light and ambient colors, have less effect. Therefore, it is a good measure for quantifying light position errors due to differences in shading. The relative mean logarithmic error (RMLE)

$$\epsilon_{\text{RMLE}} = \sqrt{\frac{1}{w_r \cdot h_r} \sum_u \sum_v \left(\log \left(\frac{\boldsymbol{I}_r^{(\text{gt})}(u,v) + 1}{\boldsymbol{I}_r^{(\text{est})}(u,v) + 1} \right) \right)^2} \tag{5.32}$$

is also a scale-independent metric suitable for similar assessments as the si-RMSE. However, its logarithmic nature makes it less sensitive to extreme differences. It provides a more nuanced measure of dissimilarity in cases where there is significant variation in intensity due to strong directional and color variations. The RGB angular error [HF04]

$$\epsilon_{\text{RGB}^\circ} = \frac{1}{w_r \cdot h_r} \sum_u \sum_v \operatorname{acos} \left(\frac{\left(\boldsymbol{I}_r^{(\text{gt})}(u,v) \right)^\mathsf{T} \boldsymbol{I}_r^{(\text{est})}(u,v)}{\left| \boldsymbol{I}_r^{(\text{gt})}(u,v) \right| \left| \boldsymbol{I}_r^{(\text{est})}(u,v) \right|} \right) \tag{5.33}$$

is better suited to measuring color differences between two images than the previous metrics. It is a good measure of how the parametric light color and ambient color differ from the spatially varying ground truth coloration.

The results for these metrics must be compared to the results of other methods for a quantitative evaluation. The state-of-the-art parametric lighting estimation method in [GHS+19] with one light source is used as a reference. The lighting representation described in [GHS+19] differs from the proposed representation in that it determines an area light source (recall Section 2.22) at a point in space with a metric surface size instead of a light direction. It also predicts light color and ambient color as well. To render image $\boldsymbol{I}_r^{(\text{est})}$ using the parameter estimation from [GHS+19], an area light must be placed in the experimental setup instead of the soft directional light. Table 5.1 shows the results of

Method	ϵ_{RMSE}	$\epsilon_{\text{si-RMSE}}$	ϵ_{RMLE}	$\epsilon_{\text{RGB}^\circ}$
[GSY+17]	0.1268	0.1708	0.078 85	3.462°
[GSH+19]	0.1178	0.1593	0.073 53	3.522°
[GHS+19](3)	0.0986	0.1323	0.061 47	2.331°
[SMT+20]	0.1459	0.1977	0.091 01	5.545°
[ZZY+21]	0.1140	0.1526	0.070 85	2.285°
[WYL+22]	0.1339	0.1798	0.082 33	3.371°
[SSS23]	0.1134	0.1530	0.070 93	3.837°
[SSS23](∗)	0.1039	0.1402	0.065 02	3.388°
[DEH+23]	0.1118	0.1504	0.070 13	3.399°

Table 5.2: Comparison by four different metrics of renderings of an experimental setup using the proposed lighting estimation ([SSS23]) versus state-of-the-art lighting estimations on a test set released in [DEH+23].

each metric averaged over the entire test data, both for images $\boldsymbol{I}_r^{(\text{est})}$ with parameters from the proposed method and with parameters from the method in [GHS+19].

In the aftermath of [SSS23], a test set[6] comprising 2240 input images along with corresponding results from several state-of-the-art lighting estimation methods has been released in [DEH+23]. The results of the lighting estimations are represented as HDR environment maps, regardless of whether the methods directly produce an environment map or a parameterized representation that can be converted into an environment map. The provided input images are excerpts (recall Section 5.4.2) from the panoramas of the Laval Indoor HDR Dataset's test set. Each of the 2240 input images has a corresponding ground truth warped panorama (recall Section 5.4), allowing for the same experimental setup to be rendered as in the previous evaluation using environment sampling for both the ground truth and predicted panoramas. Similarly, the proposed method can also be used to determine the lighting parameters and render the experimental setup with the parametric light source.

Table 5.2 presents the results of the second evaluation using the same four metrics. The test set input images have an FOV of 50°, while the proposed network is trained on input images with an FOV of 85°. Therefore, in addition to the results for the input images of

[6]The test set is publicly available at: `https://lvsn.github.io/everlight/`

the test set under "[SSS23]", the results for input images of the same view direction but with an FOV of 85° are also listed under "[SSS23](∗)".

Table 5.1 shows that the proposed method achieves comparable or even slightly better performance than the existing state-of-the-art method when considering one light source. However, the proposed method has the advantage of more stable and faster training. Furthermore, the proposed representation is less complex than the one with area lights in [GHS+19]. It is shown that comparable results can be achieved with a less complex parameterization when restricted to a single light source. A simple parametric representation is especially desirable in the case of mobile AR applications, for which the proposed method was developed. Complex light types (e.g., area lights) are not included in all frameworks for AR development or may not be supported on older devices.

Table 5.2, on the other hand, shows that the proposed method produces comparable or superior results in metrics ϵ_{RMSE}, $\epsilon_{\text{si-RMSE}}$, and ϵ_{RMLE} when compared to more complex lighting estimation methods. The compared methods encompass multiple light sources [GHS+19](3), spatially varying light descriptions through spherical basis expansion [GSH+19; SMT+20; ZZY+21], or complete environment map predictions [GSY+17; WYL+22; DEH+23]. These metrics indicate the accuracy of light positioning and resulting shadow casting. Conversely, the proposed method exhibits comparatively lower performance in metric $\epsilon_{\text{RGB}^\circ}$, which assesses color correctness. This is expected because a single light source and ambient color used in the proposed method are less accurate in approximating direction-dependent color variations in the ground truth rendering compared to multiple light sources in [GHS+19](3) or color variable representations used by the other methods.

Therefore, it is natural to address the limitations of the proposed method at this point. Simple parametric representation with a single direction of light cannot describe spatially varying lighting conditions. The estimation of a single light direction assumes an equidistant scene where the light source is positioned in the far field. In unusual cases where the light source is in the near field, a virtual object may be illuminated differently depending on its position. The simple parameterization cannot adequately describe this. Furthermore, when placing multiple virtual objects at different positions within a scene, this can also lead to confusing effects, especially when multiple dominant light sources are present in the real scene. In addition, parametric representations are unsuitable for realistically describing reflections on specular virtual objects. For specular reflections, environment map estimations, such as in [SK21; WYL+22; DEH+23], are more suitable. However, it is important to note that many compared methods are not

currently executable or real-time capable on mobile devices. Additionally, the results are not suitable for real-time shadow visualization on mobile devices, unlike the proposed method's simple parametric lighting representation.

In the previous evaluations, the contrast level o_l was neglected. It expresses the strength of the shadow cast due to the interplay between the brightness of the light source and ambient illumination of the rest of the scene. However, it is designed to work with the shadows described in Chapter 6. Therefore, an evaluation can be found in Section 6.6 as part of the evaluation of the shadow texture method.

6

Neural Shadows

The previous chapter examined the lighting conditions of the real environment. It was emphasized that a virtual object must integrate into this environment as realistically as possible for an AR application to succeed. Therefore, knowledge of the existing lighting conditions is crucial. In addition to accurately illuminating the virtual object, it is important for the object to cast realistic shadows within the scene. Figure 6.1 illustrates this concept: The virtual object in (a) lacks a sense of scene belonging due to the absence of shadow casting, giving the impression of floating above the scene. In (b), the object appears substantially more immersive due to the addition of subtle shadow casting. However, physically-based shadows, as shown in (b), are currently not directly achievable for dynamic objects on mobile or AR devices in real time. Such shadows require a large number of rays in a ray or path tracing rendering method to be accurately evaluated (recall Section 2.3). Additionally, these shadows must be dynamic to account for changes in the virtual object's position or alterations in light conditions. This remains an open problem in AR applications. Figure 6.1c demonstrates dynamic real-time shadow casting achievable with current mobile graphics engines.

(a) No shadow cast (b) Path traced shadow cast (c) Mobile engine shadow cast

Figure 6.1: A virtual squirrel is inserted into the real environment. In (a), the object is illuminated without casting a shadow. In (b), the object casts a soft shadow through non-real-time path tracing. In (c), the object casts a shadow rendered by a mobile graphics engine.

6.1 | Outline

This chapter presents a novel method for casting real-time dynamic soft shadows using neural networks. It was initially presented in [SSS23]. Section 6.2 briefly introduces shadows in general, followed by an overview of important methods for real-time shadow casting. Section 6.3 proposes a novel approach for casting shadows on planar surfaces, which is particularly suitable for AR applications where computational resources are limited. Section 6.4 describes how to create appropriate training data to train the neural network for the method, while Section 6.5 presents the network architecture and training process. Section 6.6 evaluates the method through a user study, examining its visual quality and plausibility. Finally, Section 6.7 discusses the advantages and disadvantages of the method, as well as potential use cases.

6.2 | Related Work and Background

This section begins by defining shading terminology, using a scene with a single light source as an example. The objects illuminated by this light source are called *receivers*. A point on a receiver where no light from the light source falls is considered to be in shadow. Objects that prevent light from reaching such a point on a receiver are called *occluders*. There are two types of shadows: self-shadows and cast shadows. Self-shadows occur on objects that act as both receivers and occluders (e.g., on the side of the object facing away from the light). Cast shadows are caused by an occluder on a second receiver (e.g., the shadow of an object on a flat surface).

The method described in this chapter presents a novel approach for cast shadows. Therefore, the following will focus primarily on cast shadows. Based on the previous definition of shadows, a point can either be in shadow or not. This corresponds to the hard shadows caused by infinitesimally small light sources, such as point or directional lights (recall Section 2.3.2). However, such idealized light sources do not exist in the real world. Even the sun, which is often used as a reference for directional lights, has a small angular diameter (see Section 6.4) and, therefore, does not produce completely hard shadows in the real world. Section 2.3.2 introduced area lights to describe more realistic light sources with spatial extent. This allows a point on a receiver to be partially visible from the light source, creating a partially illuminated area called *penumbra*. Points that are not visible at all from the light source lie in an area called *umbra*. Together, umbra and penumbra form the soft shadow. Figure 6.2 visualizes these regions. For a more detailed introduction to shadows, refer to [HLH+03].

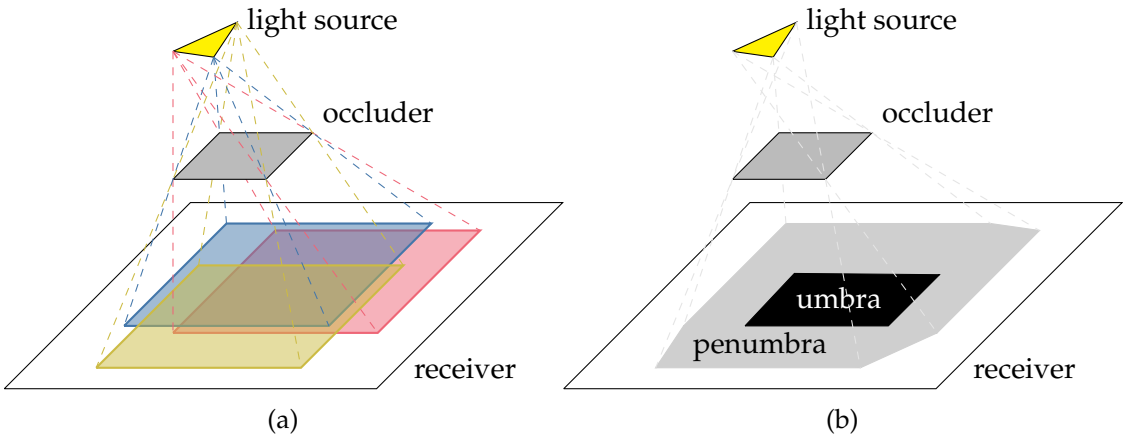


Figure 6.2: A triangular area light source casts a soft shadow of a rectangular occluder onto a planar receiver. In (a), the contributions of the individual vertices of the area light are visualized in color. In (b), the division between umbra and penumbra, together forming the entire shadow, is illustrated based on contributions from all parts of the area light source.

Physically-based rendering techniques (e.g., path tracing) simulate light transport to achieve realistic cast shadows (recall Section 2.3). However, the stochastic sampling and recursive evaluation of light paths require significant computational complexity, making path tracing unsuitable for real-time applications that require high frame rates and low latency (e.g., AR applications). Consequently, specialized algorithms have been developed for real-time rendering to efficiently approximate lighting phenomena such as shadow casting. These methods face several challenges, such as visibility determination and handling dynamic geometry, that must be addressed efficiently. Therefore, each technique has strengths and limitations, resulting in a trade-off between quality and performance. There is no one method that is superior to all others; instead, the method that best meets the application requirements must be used. In the following, a few of these methods are briefly discussed.

The pioneering works for creating shadows first focused on hard shadows. One such method is *shadow volumes* [C77], which extrude the silhouette edges of objects along the direction vector of the light source, creating a volume that represents the shadow cast by the object. Surfaces within this volume are shaded accordingly. A specialized buffer—the *stencil buffer*—masks regions to be shaded during rendering to determine which surfaces are within this volume efficiently. The shadow volumes method is continuously improved and expanded. For a comprehensive overview with further details, refer to [KS13].

Another widely adopted algorithm for real-time shadows is *shadow mapping* [W78]. This method comprises two stages. First, a depth map of the scene—also known as a *shadow map*—is rendered from the light source's perspective. Second, the scene is rendered from the camera's viewpoint. Each point visible to the camera can be transformed into the light's view coordinate system, where its visibility to the light source is tested using the shadow map. Points that are not visible to the light source are shaded. Shadow mapping techniques have also evolved since their introduction. One example is *parallel-split shadow maps* [ZSX+06], where the camera's frustum is subdivided, and a separate shadow map is rendered for each subdivision with adaptively varying resolutions. For further information on extensions and enhancements to shadow mapping methods, refer to [SWP11].

The methods previously introduced are appropriate for casting hard shadows. Techniques for casting soft shadows often extend these methods. Two types of approaches are distinguished: image-based approaches relying on shadow maps and object-based approaches relying on shadow volumes.

One of the earliest image-based techniques involves combining multiple shadow maps captured from different sample points on the surface of the light source [HH96]. Here, the penumbra region is approximated by casting shadows from different viewpoints scattered across the light source's surface and blending the resulting shadow maps. An alternative image-based method uses layered shadow maps instead of traditional shadow maps [ARH+00]. This method involves creating a separate layer for each light source and each receiver, containing information on depth and occlusion. However, these two methods are computationally expensive and require significant memory overhead. More efficient image-based approaches approximate the penumbra region of the light source by filtering the shadow map based on the depth information of the occluder geometry [F05], by convolving the shadow map with a filter kernel approximating the penumbra size [SS98; AMB+07], or by reconstructing potential occluders in world space from a baseline shadow map from the center of the light source [AHL+06].

In object-based approaches, methods that extend shadow volumes through specific heuristics, such as plateaus [H01] or smoothies [CD03], are common. Here, an additional penumbra volume is created around the hard shadow boundaries. More precise object-based methods involve calculating a penumbra volume for each edge of the occluder silhouette [AA02; AA03]. These individual penumbra volumes are combined to produce the soft shadow effect. Although these methods can produce high-quality soft shadows, they are computationally expensive. For a more detailed overview of real-time shadow

casting methods, refer to [EAS+09].

Although these methods are computationally less expensive than evaluating the rendering equation with path tracing, they still require significant computing power to create visually pleasing results. In AR applications—especially on mobile devices—computing power is a limited resource that must be shared with other computationally expensive methods. The method presented in this chapter takes a novel approach to enable visually pleasing soft shadows on such devices by encoding pre-computed shadow textures into the weights of a neural network. Therefore, it differs fundamentally from the previously described methods. While the idea of encoding images or textures in neural networks is not entirely new, this method is novel in its application to dynamic soft shadows. Compositional pattern producing networks [S07] encode image information in a network abstraction inspired by natural DNA. More recent works compress entire material textures in the form of bidirectional texture functions (BTFs) into neural networks [RJG+19; RGJ+20]. Neural radiance fields (NeRFs) [MST+21] embed entire static scenes into neural networks. In NeRFs a fully-connected neural network is trained to predict the radiance and volume density at every 3D point based on the viewing direction. This enables photorealistic and detailed view synthesis of the scene from novel perspectives. At the time of writing, NeRFs are a highly popular research field undergoing continuous improvement in training speed, inference speed, and rendering accuracy. For a comprehensive overview of NeRFs, refer to [RZ24]. A key component of NeRFs is using positional encodings. These encodings map the input 3D coordinates and viewing directions into a higher-dimensional representation using sine and cosine functions with varying frequencies. The encoding helps the network to model intricate and high-frequency details that would be difficult to represent with raw input coordinates alone. This encoding is inspired by the Transformer architecture [VSP+17], where it has already been applied to text or images [DBK+21]. Positional encoding is also used in the method presented in this chapter.

6.3 | Methodology

The previous section discussed the limitations of current dynamic real-time soft shadow methods on the constrained resources of mobile devices. However, realistic cast shadows are crucial for conveying immersion in AR applications (recall Figure 6.1). Therefore, specialized methods are necessary to deliver visually compelling results while adhering to the strict resource requirements of these devices. The presented method relies on a common scenario for AR applications: inserting a single virtual object into the real

(a) Object with cast shadow (b) Shadow texture (c) Alpha channel

Figure 6.3: An armadillo illuminated by a light source on its front-right with a cast shadow in (a). The corresponding cast shadow as a planar shadow texture in(b) with its alpha channel in (c).

environment. The object is placed on a plane estimated from the camera image. For more details on plane detection algorithms, refer to [NOB+16]. By assuming that the virtual object acts as the occluder and only the plane acts as the receiver (see Figure 6.3a); the cast shadow can be considered a texture superimposed on the otherwise transparent plane. This planar shadow texture is a 2D black texture with only alpha information (see Figure 6.3b). The alpha channel corresponds to the opacity information (recall Section 2.1.2). Therefore, an alpha value of $\alpha_{s_o} = 0$ indicates full transparency. This results in the alpha channel being the negative of the shadow texture (see Figure 6.3c). This can be confusing and challenging when rendering the textures for training in the next section using physically-based light transport simulation. A more intuitive approach is to use a single channel to describe transparency values ν_{s_t}, with $\nu_{s_t} = 0$ corresponding to full opacity. This approach results in the shadow texture, as shown in Figure 6.3b, being described by this transparency channel.

The texture's appearance depends on the geometry of the virtual object and the lighting situation of the scene. This lighting situation can be captured by a lighting estimation method, as described in Chapter 5. If a parametric representation, as described in Section 5.3, is used to characterize the lighting situation by a main light direction, the light-specific dependency is reduced to a 3D directional vector. The contributions of secondary light sources are combined into a constant ambient term. This constant illumination is a simple approximation of secondary light sources and only affects the *darkness* of the cast shadow of the virtual object. This is described by the contrast level o_l in the lighting representation, specifying the opacity with which the shadow texture should blend with the surface of the real environment. It is, therefore, multiplied by the alpha information α_{s_o} of the shadow texture. Although this method is a basic approximation

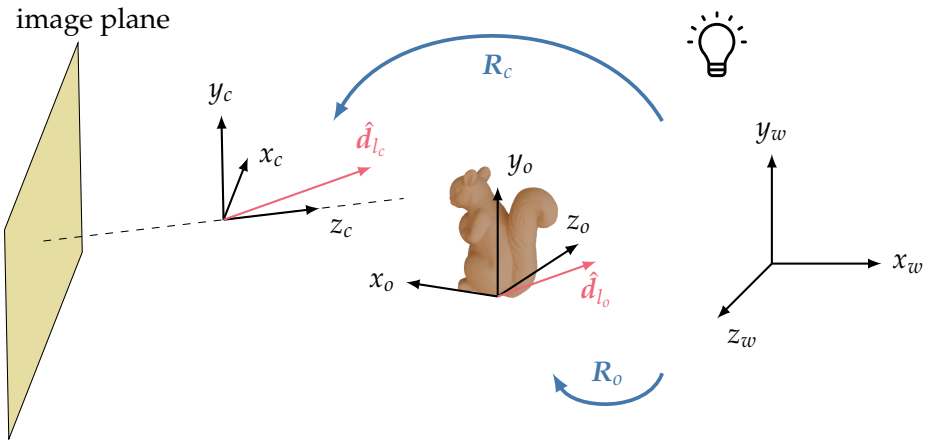


Figure 6.4: A light direction is estimated in the camera coordinate frame of the camera image. A virtual object with its own object coordinate frame is placed on a plane in the world coordinate frame. The light direction can be transformed from the camera coordinate frame to a direction relative to the object coordinate frame.

of multiple light sources in indoor scenarios, the evaluation in Section 6.6 demonstrates that it produces convincing results in many cases.

The lighting estimation method gives the light direction $\hat{\boldsymbol{d}}_l$ in the camera coordinate frame based on the current camera image. It is, therefore, denoted as $\hat{\boldsymbol{d}}_{l_c}$ in the following. A virtual object must appear invariant to camera movement at the same position in the real environment. Therefore, the plane onto which the object is placed must lie in the world coordinate frame. The two coordinate frames are connected by a rotation $\boldsymbol{R}_c$ and translation $\boldsymbol{t}_c$ (recall Section 2.2.1). The virtual object can be translated on the plane or rotated about its y-axis, resulting in an object coordinate frame. This system is also connected to the world coordinate frame by a rotation $\boldsymbol{R}_o$ and translation $\boldsymbol{t}_o$. By transforming the light direction vector $\hat{\boldsymbol{d}}_{l_c}$ into the object coordinate frame, the light direction $\hat{\boldsymbol{d}}_{l_o}$ relative to the object coordinate frame can be obtained. This eliminates the need to account for the object's orientation in the shadow texture. The relationships between the individual coordinate frames are illustrated in Figure 6.4. Since the light direction describes a directional vector—not a position—the translation between the coordinate systems can be ignored. This results in the transformation

$$\hat{\boldsymbol{d}}_{l_o} = \boldsymbol{R}_o \boldsymbol{R}_c^{-1} \hat{\boldsymbol{d}}_{l_c} \,. \tag{6.1}$$

The shadow texture for a specific object can be described by a shadow function

$$\begin{aligned} \kappa_s \quad &: \quad \mathbb{R}^5 \supset [-1,1]^5 \longrightarrow [0,1] \subset \mathbb{R}\,, \\ \kappa_s(\hat{u},\hat{v},\hat{\boldsymbol{d}}_{l_o}) &\longrightarrow \nu_{s_t} \end{aligned} \tag{6.2}$$

that assigns a transparency value ν_{s_t} to each pixel based on the relative light direction $\hat{\boldsymbol{d}}_{l_o}$ in the object coordinate frame. The domain consists of normalized pixel coordinates $\hat{u},\hat{v} \in [-1,1]$ and components of normalized directional vectors $x_{l_o}, y_{l_o}, z_{l_o} \in [-1,1]$, forming a compact subset of $\mathbb{R}^5$. If the analytic expression of this function κ_s were known, the shadow texture could be calculated quickly and efficiently. Unfortunately, an analytic solution of this function κ_s is generally unknown for arbitrary virtual objects, as shown in Section 6.4. Nevertheless, it may be approximated by a feed-forward fully connected neural network according to the universal approximation theorem [HSW89].

The theorem implies that by appropriately manipulating the weights and biases, a suitable fully connected neural network can be defined to approximate the shadow function κ_s with arbitrary accuracy. The weights and biases can be determined by training the neural network with suitable data, as explained in Section 2.4.1. he following section illustrates creating training data without solving the entire shadow function κ_s analytically.

6.4 | Training Data

Training data pairs are required to determine the weights and biases for a neural network that approximates the shadow function (6.2) of a specific object. Although the analytic solution of the shadow function is unknown, the shadow texture can be obtained for a given relative light direction $\hat{\boldsymbol{d}}_{l_o}$. The shadow function describes the transparency value ν_{s_t} of this texture for each pixel. As previously demonstrated, the transparency values correspond to the intensity of the cast shadow. Consequently, the result of the shadow function for all pixels in the texture for a specific light direction $\hat{\boldsymbol{d}}_{l_o}$ is given by a rendering of the cast shadow. From this, tuples

$$\left((\hat{u},\hat{v},\hat{\boldsymbol{d}}_{l_o}),\nu_{s_t}\right) \tag{6.3}$$

suitable as training data pairs can be derived.

An appropriate rendering scene setup is required to render shadow textures. Therefore, a suitable light source is needed to cast shadows at all. Section 2.3.2 introduced three commonly used types of light sources: point lights, directional lights, and area lights. The

first two are punctual light sources due to their infinitesimal shape. However, Section 6.3 highlighted the importance of physically correct light sources having a non-infinitesimal area to produce realistic soft shadows. Among the three types of light sources mentioned, only the area light has a non-infinitesimal shape. Nevertheless, the area light has a fixed location, resulting in a different direction vector for every point in the scene. The shadow texture, however, must be rendered for a single light direction for the entire object, as is the case with a directional light.

A hybrid light source between directional and area light can be defined, hereafter referred to as *soft directional light*. Therefore, the angular diameter γ_{ad} commonly used in astronomy can be employed [K20]. It describes the size of a celestial object by the solid angle it occupies in the night sky. The object's actual size can be calculated if the distance to it is known. This principle can be applied to the light transport simulation in rendering, where all incident radiance contributions from all incoming directions $\hat{\boldsymbol{\omega}}_i$ of the hemisphere Ω surrounding a point $\boldsymbol{x}$ are accumulated (recall Section 2.3.1). A soft directional light can be defined as an object that occupies an angular diameter on this hemisphere, similar to a celestial object in the sky. Therefore, it possesses a circular area on the hemispherical surface. An angular diameter of $\gamma_{\text{ad}} = 20^\circ$ is used for all results in this chapter. In comparison, the sun's angular diameter in the sky is $\gamma_{\text{ad}} \approx 0.5^\circ$.

The shadow texture represents the interaction between the receiver and occluder based on this single light source. The direct lighting contribution of this one light source with emitted radiance $L_e^{(e)}$ results in a total radiant power per unit area, *irradiance*

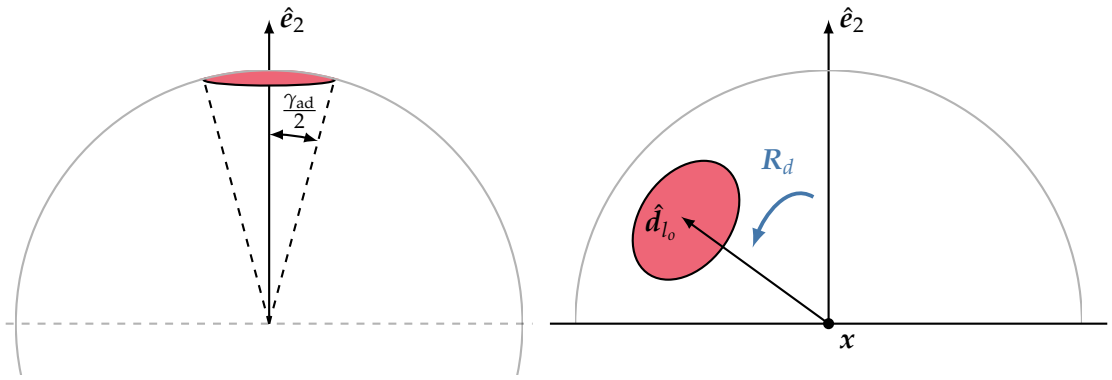
$$E^{(e)}(\boldsymbol{x}) = \int_\Omega L_e^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_i)(-\hat{\boldsymbol{\omega}}_i^\mathsf{T} \hat{\boldsymbol{n}}_x)\mathrm{d}\hat{\boldsymbol{\omega}}_i\,, \tag{6.4}$$

for a point $\boldsymbol{x}$ with surface normal $\hat{\boldsymbol{n}}_x$. As explained in Section 2.3.2, it is often more convenient to use the area formulation

$$E^{(e)}(\boldsymbol{x}) = \int_A L_e^{(e)}(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_i) \frac{(-\hat{\boldsymbol{\omega}}_y^\mathsf{T} \hat{\boldsymbol{n}}_x)(\hat{\boldsymbol{\omega}}_y^\mathsf{T} \hat{\boldsymbol{n}}_y)}{|\boldsymbol{x} - \boldsymbol{y}|^2} \Theta_v(\boldsymbol{x}, \hat{\boldsymbol{\omega}}_y)\mathrm{d}A \tag{6.5}$$

instead of the hemispherical integral, so the accumulation of incident radiance can be expressed as an integration on the surface A of the light source. The visibility function Θ_v evaluates whether a point $\boldsymbol{y}$ on the surface A with surface normal $\hat{\boldsymbol{n}}_y$ is directly visible from point $\boldsymbol{x}$ through the normalized direction $\hat{\boldsymbol{\omega}}_y$ from $\boldsymbol{y}$ to $\boldsymbol{x}$.

The surface points $\boldsymbol{y}$ of the soft directional light are on a circular area around $\hat{\boldsymbol{d}}_{l_o}$ on the



(a) (b)

Figure 6.5: The circular area on the spherical surface between the up vector $\hat{e}_2$ and a polar angle $\gamma_{ad}/2$ in (a) is rotated in (b) using a rotation R_d so that its center lies around $\hat{d}_{l_o}$.

unit hemisphere. A circular area on a unit sphere can be calculated by the surface integral

$$A = \int_A \mathrm{d}A = \int_0^{2\pi} \int_0^{\gamma_{ad}/2} \sin\theta \, \mathrm{d}\phi \, \mathrm{d}\theta \,, \tag{6.6}$$

assuming is symmetric about the up vector (see Figure 6.5a). To apply this parametrization to a circular area around an arbitrary point $\hat{d}_{l_o}$ on the unit sphere surface (see Figure 6.5b), a coordinate transformation described by the rotation matrix R_d between $\hat{d}_{l_o}$ and the up vector $\hat{e}_2$ is necessary. Therefore, a parameterized point

$$y = x + R_d \begin{pmatrix} \cos\phi \sin\theta \\ \cos\theta \\ \sin\phi \sin\theta \end{pmatrix} =: x + \hat{r}_l(\hat{d}_{l_o}, \phi, \theta) \tag{6.7}$$

on the circular area around $\hat{d}_{l_o}$ on the unit hemisphere can be described. This leads to:

$$\hat{\omega}_y = \frac{y - x}{|y - x|} = \hat{r}_l(\hat{d}_{l_o}, \phi, \theta) \,, \tag{6.8}$$

since

$$|y - x| = \left| \hat{r}_l(\hat{d}_{l_o}, \phi, \theta) \right| = 1 \,. \tag{6.9}$$

The radiance emitted by the soft directional light is constant throughout the scene, similar to the one emitted by a directional light (recall Section 2.3.2). Therefore, the term can be

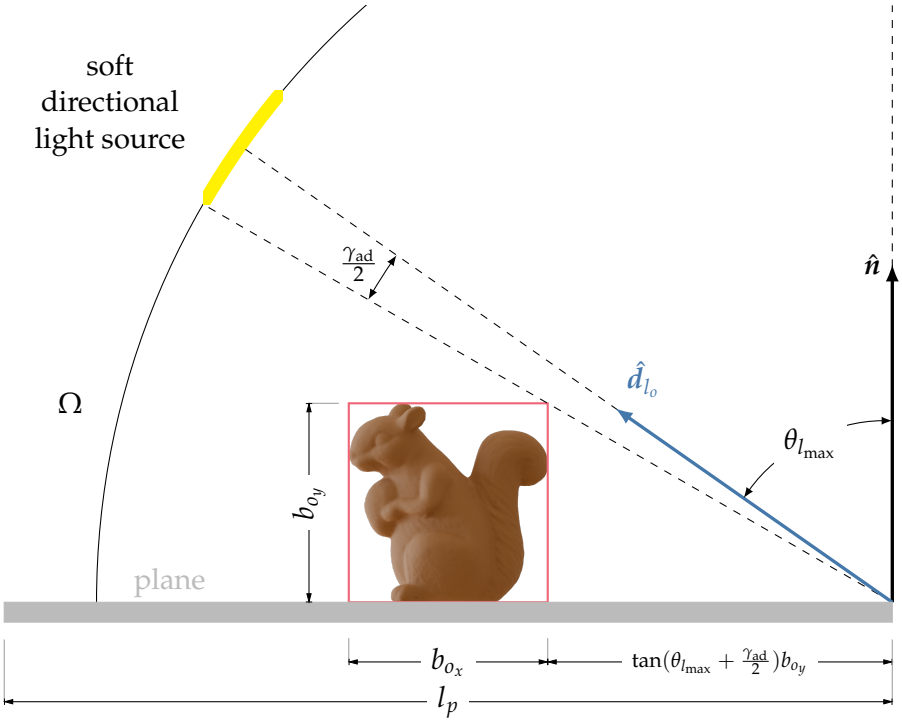


Figure 6.6: The geometric relationship between the size of the bounding box, here generalized for $\max(b_{o_x}, b_{o_z}) = b_{o_x}$, and the resulting requirements for the size of the plane l_p to receive the entire shadow cast.

factored out of the integral of Equation (6.5), resulting in:

$$E^{(e)}(\boldsymbol{x}) = L_e^{(e)} \int_0^{\gamma_{ad}/2} \int_0^{2\pi} (-\hat{\boldsymbol{r}}_l^{\mathsf{T}} \hat{\boldsymbol{n}}_x)(\hat{\boldsymbol{r}}_l^{\mathsf{T}}(-\hat{\boldsymbol{d}}_{l_o}))\Theta_v(\boldsymbol{x}, \hat{\boldsymbol{r}}_l) \sin\theta \, \mathrm{d}\phi \, \mathrm{d}\theta \,. \tag{6.10}$$

In this scene setup, the virtual object acts as the occluder for the light source, and a texture plane below the object as the receiver for the cast shadow. The texture plane should be dimensioned to receive the entire cast shadow for all possible light directions $\hat{\boldsymbol{d}}_{l_o}$. For light directions near the horizon, this leads to long shadows and, consequently, large planes. Therefore, it is advisable to limit the possible light directions by defining a maximum angle $\theta_{l_{max}}$ between the up vector and the light direction $\hat{\boldsymbol{d}}_{l_o}$. For the results in this thesis, the light directions are limited to $\theta_{l_{max}} = 45°$. The occluder is placed at the center of the square texture plane to ensure that the plane's position can remain constant for different light directions. The minimum required size

$$l_p = \max(b_{o_x}, b_{o_z}) + 2\tan\left(\theta_{l_{max}} + \frac{\gamma_{ad}}{2}\right) b_{o_y} \tag{6.11}$$

of the texture plane can be determined based on the occluder's bounding box $\mathcal{B}_o$ (recall Equation (3.24)) and the geometric relations illustrated in Figure 6.6.

The texture plane can act as an image sensor, when rendering the shadow texture. The irradiance $E^{(e)}$ at each point $\boldsymbol{x}$ on the plane (see Equation (6.10)) corresponds to the amount of light received per second per square meter by the sensor at that point. In a real camera, the amount of light reaching the sensor during the exposure time t_H is measured as radiant exposure

$$H^{(e)}(\boldsymbol{x}) = \int_0^{t_H} E^{(e)}(\boldsymbol{x}) \mathrm{d}t \,. \tag{6.12}$$

The camera's nonlinear response function converts the exposure $H^{(e)}(\boldsymbol{x})$ to intensity $I(\boldsymbol{x})$ to match human perception (recall Section 5.2). However, the shadow texture is not intended to be perceived as a natural image; it should record the shadow as a transparency channel. Consequently, the response function may be the linear identity function, which results in: $I(\boldsymbol{x}) = H^{(e)}(\boldsymbol{x})$. In a real camera, the exposure time t_H is set to produce a neutrally exposed image. For the shadow texture, areas outside the shadow should have an intensity of 1 since they correspond to full transparency $\nu_{s_t} = 1$. The emitted radiance of the light source $L_e^{(e)}$ is constant. The surface normal of the plane is always pointing upwards $\hat{\boldsymbol{n}}_x = \hat{\boldsymbol{e}}_2$. The spatial dependence in $\boldsymbol{x}$ in Equation (6.10) depends only on the visibility term $\Theta_v(\boldsymbol{x}, \hat{\boldsymbol{r}}_l)$. Therefore, for an area outside the shadow (i.e., $\Theta_v = 1$) the irradiance $E^{(e)}$ is also constant. Consequently, the two remaining variables that influence the intensity are the exposure time t_H and the emitted radiance of the light source $L_e^{(e)}$. Since all variables are time-independent, it follows: $H^{(e)} = E^{(e)} t_H$. Setting the exposure time to a constant (e.g., $t_H = 1$) and analytically solving the integral in Equation (6.10) with $\Theta_v = 1$ for a given light direction $\hat{\boldsymbol{d}}_{l_o}$ leads to an exposure $H^{(e)}$ only depending on the emitted radiance $L_e^{(e)}$ of the light source. Therefore, $L_e^{(e)}$ can be used as a scaling factor so that $H^{(e)} = 1$.

When evaluating the irradiance $E^{(e)}$ for scenes with simple occluder geometries (e.g., a box), the visibility term Θ_v can be expressed as an analytically solvable function. However, the visibility term cannot be expressed analytically for more complex geometries. In this case, the irradiance cannot be calculated analytically. The entire rendering equation can be evaluated numerically using path tracing (recall Section 2.3.3). A similar approach can be used here to calculate the irradiance on the plane. In this case, Equation (6.10) is numerically approximated using Monte Carlo integration [R81] to calculate the surface integral. Monte Carlo integration involves randomly sampling the integrand function and averaging the results to approximate the integral value. Each sample must have an

equal probability of being chosen during Monte Carlo integration to ensure unbiased results. Therefore, it is necessary to uniformly sample the integral domain, which in this case is the surface of the light source. Section 3.3.3 described a detailed method for uniformly sampling surfaces of objects with arbitrary shapes. In the case of a unit sphere, a uniformly distributed sample

$$\theta_s = \arccos\left(1 - 2\rho_{\mathrm{rnd}_1}\right), \tag{6.13}$$

$$\phi_s = 2\pi\rho_{\mathrm{rnd}_2} \tag{6.14}$$

can be generated using two uniform random variables $\rho_{\mathrm{rnd}_1}, \rho_{\mathrm{rnd}_2} \in [0,1]$. Therefore, for the circular area of the soft directional light on the unit hemisphere, a uniformly distributed sample is given by

$$\theta_s = \arccos\left(1 - \left(1 - \cos\frac{\gamma_{\mathrm{ad}}}{2}\right)\rho_{\mathrm{rnd}_1}\right) \tag{6.15}$$

and (6.14). The set $\mathbb{S}_l$, with $|\mathbb{S}_l|$ such samples, forms the sample set of the Monte Carlo integration. The integrand

$$((\hat{\boldsymbol{r}}_l(\theta_s,\phi_s))^{\mathsf{T}}\,\hat{\boldsymbol{e}}_2)((\hat{\boldsymbol{r}}_l(\theta_s,\phi_s))^{\mathsf{T}}\,\hat{\boldsymbol{d}}_{l_o})\Theta_v(\boldsymbol{x},\hat{\boldsymbol{r}}_l(\theta_s,\phi_s))\sin\theta_s \tag{6.16}$$

of Equation (6.10) can be evaluated for each sample $(\theta_s,\phi_s) \in \mathbb{S}_l$. To determine the visibility term $\Theta_v(\boldsymbol{x},\hat{\boldsymbol{r}}_l)$, it is necessary to check if the path from point $\boldsymbol{x}$ along $\hat{\boldsymbol{r}}_l$ intersects the occluder, as described in Section 3.4.1.

Therefore, the irradiance at point $\boldsymbol{x}$ can be approximated by:

$$E^{(e)}(\boldsymbol{x}) \approx \frac{L_e^{(e)}}{|\mathbb{S}_l|}\sum_{(\theta_s,\phi_s)}^{|\mathbb{S}_l|}((\hat{\boldsymbol{r}}_l(\theta_s,\phi_s))^{\mathsf{T}}\,\hat{\boldsymbol{e}}_2)((\hat{\boldsymbol{r}}_l(\theta_s,\phi_s))^{\mathsf{T}}\,\hat{\boldsymbol{d}}_{l_o})\Theta_v(\boldsymbol{x},\hat{\boldsymbol{r}}_l(\theta_s,\phi_s))\sin\theta_s \tag{6.17}$$

using Monte Carlo integration. The irradiance allows for the calculation of intensity at each point $\boldsymbol{x}$. However, sensor pixels (u,v) do not correspond to infinitesimal points but have spatial extent. To the compute pixel intensity $I(u,v)$, the intensity $I(\boldsymbol{x})$ must be integrated over the area of the pixel. Since computing the intensity at an infinite number of points $\boldsymbol{x}$ is impractical, the plane or sensor surface can be uniformly sampled with a set of points $\mathbb{S}_p$. This yields the pixel intensity

$$I(u,v) = \frac{1}{|\mathbb{S}_p|^{(u,v)}}\sum_{i=1}^{|\mathbb{S}_p|^{(u,v)}} I(\boldsymbol{x}_{s_i}), \tag{6.18}$$

where $|\mathbb{S}_p|^{(u,v)}$ corresponds to the number of samples within this pixel. This grayscale pixel intensity $I(u,v)$ corresponds to the transparency value ν_{s_t} for pixel (u,v).

Figure 6.7: Multiple shadow textures for an armadillo (see Figure 6.3a) for different light directions. [Image sourced from [SSS23].]

This process yields a shadow texture (i.e., a square image $\boldsymbol{I}_s$ with w_s^2 pixels) for a given object and light direction $\hat{\boldsymbol{d}}_{l_o}$. Figure 6.7 shows an example of several shadow textures of the same object from different light directions. By rendering textures for N_{d_l} different light directions, a set of

$$N_{\text{td}} = N_{d_l} w_s^2 \tag{6.19}$$

training data pairs is compiled. In deep learning, a rule of thumb, suggesting the training data size should be approximately ten times the number of network parameters [AvC18], is often applied to determine the optimal training data size N_{td}. However, for the shadow textures, networks with a relatively small number of parameters are used, for which such an amount of data does not cover a sufficient number of light directions N_{d_l}. Figure 6.8 illustrates networks with different numbers of parameters trained with datasets of varying sizes. For a test set comprising shadow textures $\boldsymbol{I}_s$ for 1000 light directions $\hat{\boldsymbol{d}}_{l_o}$, the ground truth pixel values $\boldsymbol{I}_s^{(\text{gt})}(u, v)$ are compared with the results $v_{s_t}^{(\text{est})}$ of each network using the average RMSE ϵ_{RMSE} (recall Equation (5.30)). Saturation occurs for a training data size N_{td} between 13 million and 20 million. For a resolution of 256×256, this corresponds to between 200 and 300 shadow textures. Given that the data is theoretically free of noise and that additional data do not increase the complexity of the problem, it can be concluded that the capability of the network to generalize is not affected by additional data. For the results in Section 6.6, a number of $N_{d_l} = 301$ shadow textures at a resolution of 256×256 were used.

For a well-suited training dataset, the range of possible light directions, in unit spherical coordinates $\theta \in [0, \theta_{l_{\max}}]$ and $\phi \in [0, 2\pi]$, must be uniformly covered by the different

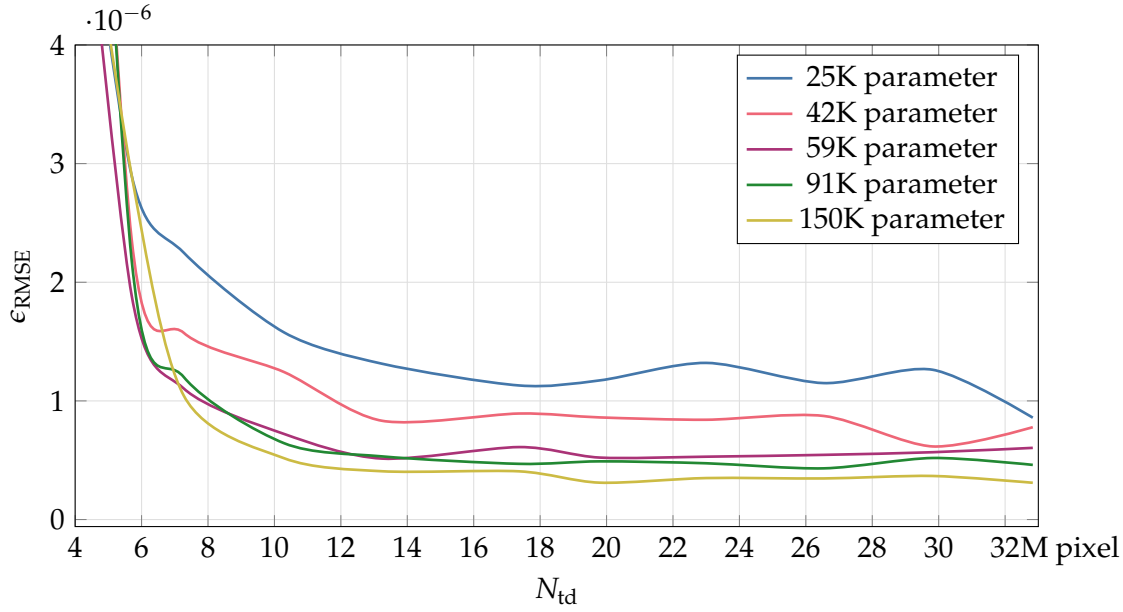


Figure 6.8: The average RMSE ϵ_{RMSE} between the shadow texture generated by neural networks with different numbers of parameters (see Table 6.1 for network configurations), trained on datasets of different sizes N_{td}, and the ground truth shadow texture for 1000 light directions $\hat{\boldsymbol{d}}_{l_o}$.

Algorithm 6: Generating training data pairs for neural shadows

Data: Virtual object geometry
Result: Set of training data pairs $((\hat{u}, \hat{v}, \hat{\boldsymbol{d}}_{l_o}), \nu_{s_t})$

generate N_{d_l} light directions $\hat{\boldsymbol{d}}_{l_o}$
forall $\hat{\boldsymbol{d}}_{l_o}$ **do**
  uniformly sample plane with points $\boldsymbol{x}_s$
  uniformly sample light surface with points $\boldsymbol{y}$
  forall pixel (u, v) in $\boldsymbol{I}_s$ for $\hat{\boldsymbol{d}}_{l_o}$ **do**
    $\nu_{s_t} \leftarrow 0$
    forall points $\boldsymbol{x}_s$ in pixel (u, v) **do**
      forall points $\boldsymbol{y}$ **do**
        form ray $\boldsymbol{r}_l$ from $\boldsymbol{x}_s$ to $\boldsymbol{y}$
        if $\Theta_v(\boldsymbol{x}_s, \hat{\boldsymbol{r}}_l) = 1$ **then**
          $\nu_{s_t} \leftarrow \nu_{s_t} + \frac{L_e^{(e)}}{|\mathrm{S}_p|^{(u,v)}|\mathrm{S}_l|}(\hat{\boldsymbol{r}}_l^{\mathsf{T}}\hat{\boldsymbol{e}}_2)(\hat{\boldsymbol{r}}_l^{\mathsf{T}}\hat{\boldsymbol{d}}_{l_o})\sin\theta_s$
    calculate normalized pixel coordinate $(\hat{u}, \hat{v})$ for pixel (u, v)
    save training data pair $((\hat{u}, \hat{v}, \hat{\boldsymbol{d}}_{l_o}), \nu_{s_t})$

shadow textures. The required number of shadow textures N_{d_l} for training the neural network is too small to have statistical significance. Consequently, random sampling cannot guarantee uniform coverage. Section 3.3.2 showed how sunflower growth patterns projected on spheres can be used to generate a deterministic uniform sampling of points on the surface of the unit sphere, respectively, the portion of the surface corresponding to the range of possible light directions. The entire procedure to generate the training data is illustrated in Algorithm 6 in pseudocode.

6.5 | Neural Network

The previous section described a dataset comprising shadow textures. Pixels of the shadow texture correspond to function values of the shadow function. The dataset is intended to train a suitable neural network, approximating the shadow function as accurately as possible while being small and compact enough to be efficiently interfered on mobile devices in real time.

6.5.1 | Network Architecture

Fully connected neural networks are universal function approximators and, therefore, suitable for theoretically approximating the shadow function with arbitrary precision (recall Section 6.3). The shadow function maps a 5D input space consisting of a normalized pixel coordinate $(\hat{u}, \hat{v})$ and light direction $\hat{\boldsymbol{d}}_{l_o}$ to a 1D transparency value ν_{st}. Such a low-dimensional input space amplifies the tendency of neural networks to be biased towards lower frequency functions [RBA+19]. This results in the network being unable to express the high-frequency details in shadow textures, leading to a poor representation (see Figure 6.9c). Mapping the input into a higher-dimensional space before passing it into the network allows the network to capture high-frequency variations better, as shown in [RBA+19]. This enables the network to learn more complex relationships between input and transparency value. Figure 6.9 illustrates the significance of the input mapping, showing in (b) the resulting shadow texture of a network with an input mapping (see Equation (6.20)) and in (c) the resulting shadow texture of a network without the input mapping. The networks have otherwise identical architectures. In comparison, the ground truth rendered shadow texture is shown in Figure 6.9a.

(a) ground truth (b) with input mapping (c) without input mapping

Figure 6.9: A comparison between a rendered ground truth shadow texture in (a), the shadow texture generated by a neural network with input mapping in (b), and the shadow texture of an otherwise identical neural network without input mapping in (c).

A typical type of mapping comprises a set of periodic functions with different frequencies [TSM+20]. For each input parameter χ in the shadow function, an encoding function

$$\begin{aligned} &\Phi_{\text{PE}} \quad : \quad \mathbb{R} \longrightarrow \mathbb{R}^{2N_{\text{pe}}+1}, \\ &\Phi_{\text{PE}}(\chi) = \left(\chi, \sin\left(2^0\pi\chi\right), \cos\left(2^0\pi\chi\right), \ldots, \sin\left(2^{N_{\text{pe}}-1}\pi\chi\right), \cos\left(2^{N_{\text{pe}}-1}\pi\chi\right)\right) \end{aligned} \tag{6.20}$$

consisting of the original value and a higher-dimensional sequence of N_{pe} alternating sine and cosine functions is used, similar to positional encoding in Transformers [VSP+17]. The purpose of this encoding differs significantly from that in Transformers, where it is used to give the input sequence an order. Here, the encoding Φ_{PE} is used to map the input coordinates into a higher-dimensional space to approximate higher frequency functions, analogous to its use in NeRFs. A different encoding dimensionality is used for each input type. For the normalized image space $\in [-1, 1]$, an encoding with $N_{\text{pe}} = 10$ sine and cosine functions is used. For the light direction vector $\hat{\boldsymbol{d}}_{l_o}$, an encoding with $N_{\text{pe}} = 4$ sine and cosine functions is used analogously to the encoding of the viewing vector in [MST+21]. This results in an overall mapping from the input space $\subset \mathbb{R}^5$ to a higher-dimensional space $\subset \mathbb{R}^{69}$.

The neural network receiving the encoding vector as input is fully connected. It comprises N_{l_t} hidden layers, each with a constant number of N_{ls} neurons per layer. Each neuron in the hidden layer is ReLU activated (recall Equation (2.26)). The network output is scaled using a sigmoid function (recall Equation (2.28)) to ensure that the transparency value lies within $[0, 1]$. Table 6.1 contains all the tested network configurations and the resulting number of network parameters. Furthermore, the network configurations shown in Figure 6.8 are highlighted in color accordingly. Tested configurations with fewer

N_h \ N_{ls}	**96**	**128**	**196**	**256**
2	25 441	42 113	91 141	149 761
3	34 753	58 625	129 753	-
4	44 065	75 137	-	-

Table 6.1: The number of total parameters of network configurations with different numbers of hidden layers N_h and layer sizes N_{ls}. The highlighted cells are color-coded to match the corresponding curves in Figure 6.8.

parameters could not capture sufficient information to converge, and configurations with more parameters exhibited increasingly unstable training and, consequently, often failed to converge. Furthermore, deeper networks with $N_h > 3$ become increasingly challenging to train. Therefore, the recommendation is to use configurations with $N_h = 2$ or $N_h = 3$.

6.5.2 | Training Procedure

When training the fully connected neural network for a specific object, the weights and biases of the network are optimized using the training data described in Section 6.4. A forward pass through the network should approximate the shadow function for this object as accurately as possible. The photometric loss, given by the MSE

$$\mathcal{L}_{\text{MSE}}(\nu_{s_t}^{(\text{est})}, \nu_{s_t}^{(\text{gt})}) \tag{6.21}$$

between the ground truth pixel transparency $\nu_{s_t}^{(\text{gt})}$ and the network estimate $\nu_{s_t}^{(\text{est})}$, is an appropriate choice for the loss function. The number of network parameters is relatively small compared to that of typical neural networks, and all training data fits into the memory of the GPU. Therefore, it is possible to omit mini-batches during training and instead perform the optimization on the entire dataset. This has the advantage of significantly reducing the duration of training by eliminating the loading of mini-batches into GPU memory. By eliminating mini-batches, optimization is done using ordinary gradient descent instead of stochastic gradient descent. One optimization step includes all N_{td} training samples and, therefore, corresponds to one epoch. The network is trained for 10 000 epochs, based on empirical observations indicating that the network typically converges by this number of iterations. The initial learning rate $\lambda_{\text{lr},0} = 0.001$ is exponentially reduced per epoch k by:

$$\lambda_{\text{lr}}(k) = 0.1^{\frac{k}{10\,000}} \lambda_{\text{lr},0} \tag{6.22}$$

to one-tenth of its original value after the 10 000th epoch. Training is completed in less than five minutes on an Nvidia RTX A6000.

6.6 | Evaluation

The overall quality of the neural shadow method can be assessed by a qualitative evaluation of the entire pipeline, including the lighting estimation described in Chapter 5. Both methods—in combination—aim to provide users with an immersive experience when inserting virtual objects in the real environment in an AR context. Therefore, a user study is conducted to examine the overall quality of the insertion.

For this evaluation, new HDR panoramas[1] are used from outside the dataset used for training the lighting estimation. For each panorama, a rectilinear image excerpt (recall Section 5.4.2) is projected, serving as the equivalent of the camera image in an AR application. Using the lighting estimation method, the individual parametric lighting parameters—light direction $\hat{\boldsymbol{d}}_{l_c}$, light color $\boldsymbol{c}_l$, ambient color $\boldsymbol{a}_l$, and contrast level o_l—are estimated for this image excerpt. A virtual object is placed in the image excerpt and illuminated by a directional light with a light color $\boldsymbol{c}_l$ and light direction $\hat{\boldsymbol{d}}_{l_c}$ relative to the camera coordinate system, in combination with a constant ambient illumination with color $\boldsymbol{a}_l$. Following the size of the object's bounding box, a square plane—referred to as *texture plane*—with a side length l_p (recall Equation (6.11)) is placed beneath the object. The neural network described in Section 6.5 estimates the shadow texture of the object for the light direction $\hat{\boldsymbol{d}}_{l_o}$ in the object coordinate frame. The alpha channel is multiplied by the contrast level o_l. This texture is assigned to the texture plane. The complete configuration is illustrated in Figure 6.10. The resulting images serve as simulated test cases of an AR application and are referred to below as *generated images*.

For each generated image, a reference image is needed as a ground truth. Therefore, the same image excerpt is used as the background, and the virtual object is inserted at the same position. Here, a plane is again positioned beneath the object, used as a shadow catcher. The original HDR panorama is modified using the warping operation described in Section 5.4.3 to resemble a light probe at the insertion point. The warped panorama is then used to realistically illuminate the virtual object and simulate its actual shadow cast on the plane through path tracing with environment sampling (recall Section 2.3.3).

The evaluation set comprises 20 image pairs. Figure 6.11 presents a sample of three

[1] Panoramas sourced from: `https://polyhaven.com/hdris/indoor`

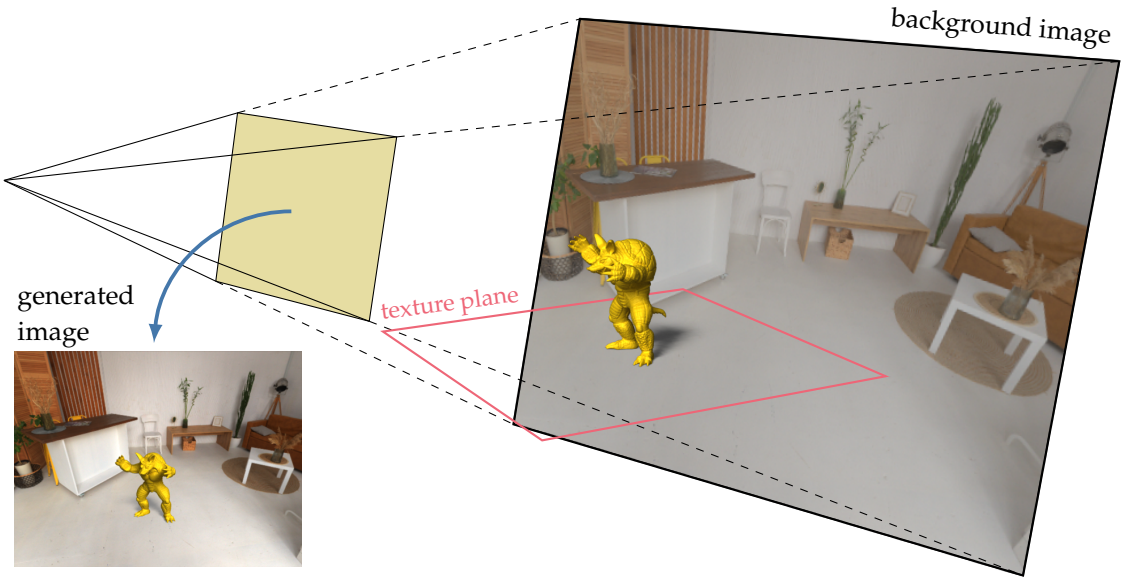


Figure 6.10: The setup to render a *generated image* consisting of a background image, a virtual object (Armadillo), and a shadow texture on the texture plane below the object.

image pairs. The user study initially conducted for [SSS23]presented these images to 50 participants. In an initial survey, participants were asked to assess how realistically the inserted object integrates into the existing scene regarding illumination and shadow cast. The participants were explicitly instructed to ignore other factors, such as size ratios, object selection, context, and style when making their judgments. For each participant, ten images were randomly chosen from the generated images, and the remaining ten images were from the reference images. A Likert scale [L32] was used as the rating scheme, with responses ranging from 1 (indicating a high degree of unrealism) to 5 (indicating a high degree of realism).

In a subsequent survey, participants were presented with the 20 image pairs side by side. Participants were asked to decide in which of the two images the virtual object was more plausibly inserted into the existing scene. The optimal outcome for this type of evaluation would be if the participants cannot decide which of the two images is more plausible at a rate better than chance. This phenomenon is referred to as *perfect confusion*. It occurs when the proportion of participants who prefer the generated image is equal to that of participants who prefer the reference image. Conversely, if a high percentage of participants prefer the reference image, the quality of the proposed method would be perceptibly worse. On the other hand, if a high percentage of participants prefer the

(a) Reference image (b) Generated image

Figure 6.11: Three pairs of images from the set used for the user study. The reference images (a) rendered with path tracing and environment sampling. The generated images (b) rendered with direct lighting using a directional light, constant ambient illumination, and the neural shadow texture.

	Reference	**Generated**
Rating	3.49 ± 0.38	3.26 ± 0.46
Votes	0.544%	0.456%

Table 6.2: User study results for 20 generated and reference image pairs (50 participants). Ratings indicate the perceived realism of how well an object fits into the scene, focusing solely on illumination and shadow casting, on a scale from 1 (very unrealistic) to 5 (very realistic). Votes represent the percentage preference for image groups regarding plausibility (50% = perfect confusion).

generated image, it would indicate that something anomalous is present in the study. Potential reasons could include artifacts in the reference images that make them appear implausible, a bias in the generated images making them appear beautified, or simply a flawed study interface design that influences participants' responses in unintended ways.

Table 6.2 presents the results of the two surveys conducted as the user study for [SSS23]. In the first survey, participants rated the realism of the reference images only slightly higher than the generated images. In the second survey, nearly as many participants considered the generated images more plausible than those who preferred the reference image. The result is close to perfect confusion, indicating that the proposed method effectively conveys a sense of immersion in users when embedding virtual objects into the real environment in AR applications.

6.7 | Discussion

The novel method presented in this chapter aims to improve the quality of real-time capable cast shadows in AR applications. The traditional rendering techniques for shadow casting, as outlined in Section 6.2, often require considerable computational resources and processing time, making them less suitable for real-time applications on many AR devices. The key innovation of the proposed method lies in the use of neural networks to encode shadow textures, which offers a significant speed advantage while maintaining high-quality visual results. The preceding section's evaluation supports the claim that these shadow textures are visually plausible and effectively contribute to the realism of virtual objects integrated into real-world scenes. A further empirical evaluation on a Google Pixel 8 indicates that the inference time for creating a 256×256 shadow texture on a mobile device is approximately 17 ms for the network with 42K parameters. It should be noted that the shadow texture is only recalculated when the object's pose or lighting situation changes, in contrast to rendered shadows that must be updated every frame.

Despite the advantages, the proposed method has certain limitations that must be addressed. One limitation is that a neural network must be trained separately for each virtual object. A certain amount of preprocessing is required to create the necessary training data and to conduct the training itself, which must occur on a more computationally powerful unit (e.g., a server). However, the entire process can be fully automated and completed within minutes. Nevertheless, it is not possible to dynamically introduce previously unknown virtual objects on the fly. Furthermore, the current method is limited to producing cast shadows on a planar ground surface. It does not support shadow casting on other objects or walls within the scene, which limits its applicability in more complex environments.

Therefore, the method is particularly suited for use cases where a known virtual object is to be inserted into an otherwise real environment on a flat surface. One potential application includes AR scenarios in the furniture retail industry, where a piece of furniture is digitally placed into a room for preview purposes through AR.

7

Conclusion

This final chapter summarizes the individual contributions of this thesis, which explored different aspects of the MR spectrum.

In physically-based animation, a two-stage granular material simulation method was introduced. This method allows the real-time simulation of significantly larger numbers of particles compared to existing approaches. The proposed method achieves a computational speedup of up to two orders of magnitude over state-of-the-art techniques, allowing for the real-time computation of up to half a million particles. While the accuracy of the simulation does not match that of simulations used for engineering applications, it is well-suited for plausible visualization.

To improve photometric registration, a lighting representation was presented to characterize the lighting conditions using a minimal set of parameters. A neural network was trained to estimate these parameters from a single RGB camera image. The results showed that this lighting estimation method requires fewer parameters than existing state-of-the-art approaches while producing comparable results.

A second key contribution to photometric registration was developing a novel method for casting shadows of virtual objects. This innovative approach creates high-quality soft shadows with minimal computational cost. A tiny neural network was trained for each object to generate a shadow texture based on the main direction of light. The network is small enough to be efficiently interfered in real time on AR devices with limited resources. In addition, the network training process is lightweight, making this method practical for commercial applications.

This thesis's contributions provide new insights and advances in MR, computer vision, computer graphics, and computational simulation. Through innovative approaches inspired by other research areas, time-critical operations have been accelerated. Moreover, a contribution to the improvement of photometric registration in AR has been made.

A

Formula Derivation

A.1 | Discrete Points on the Fermat Spiral

At this point, the discrete points on the Fermat spiral are derived in detail, starting with a set of polar coordinates (r, ϕ). The angular spacing between two elements corresponds to the golden angle Φ_γ, therefore:

$$\phi_i = \Phi_\gamma \, i \, . \tag{A.1}$$

Since the radius of the generative spiral should increase proportionally to the square root of the spiral angle ϕ [V79], this leads directly to:

$$r_i = \sqrt{\phi_i} = \sqrt{\Phi_\gamma \, i} = \sqrt{\Phi_\gamma} \cdot \sqrt{i} \, . \tag{A.2}$$

Since the golden angle is a constant, it can be incorporated into a general proportionality constant α, yielding:

$$r_i = \alpha \sqrt{i} \, . \tag{A.3}$$

Let r_{dsk} denote the total radius of the disk. If the $N_{\text{S}_{\text{dsk}}}$-th sample lies on the edge of the disk, it follows that:

$$r_{\text{dsk}} = \alpha \sqrt{N_{\text{S}_{\text{dsk}}}} \, , \tag{A.4}$$

which leads to:

$$\alpha = \frac{r_{\text{dsk}}}{\sqrt{N_{\text{S}_{\text{dsk}}}}} \, . \tag{A.5}$$

As demonstrated in [SJ06], a shift by $1/2$ achieves a more balanced uniformity at the disk center, thus resulting in:

$$r_i = \alpha \sqrt{i - \frac{1}{2}} \, , \tag{A.6}$$

finally leading to:

$$r_i = \frac{r_{\text{dsk}}}{\sqrt{N_{\text{S}_{\text{dsk}}} - \frac{1}{2}}} \sqrt{i - \frac{1}{2}} \, . \tag{A.7}$$

A.2 | Non-penetration Constraint Gradients

In the following, the $*$ in the superscript is omitted for readability. At this point, the gradients of the non-penetration constraint

$$c_{\text{npt}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left|\boldsymbol{x}_{ij}\right| - 2r_{lr} \tag{A.8}$$

will be derived in detail with

$$\boldsymbol{x}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i \tag{A.9}$$

and

$$\left|\boldsymbol{x}_{ij}\right| = \sqrt{(\boldsymbol{x}_j - \boldsymbol{x}_i)^\mathsf{T}(\boldsymbol{x}_j - \boldsymbol{x}_i)}\,. \tag{A.10}$$

First, the gradient

$$\underset{\boldsymbol{x}_i}{\nabla} c_{\text{npt}} = \underset{\boldsymbol{x}_i}{\nabla} \left(\left|\boldsymbol{x}_{ij}\right| - 2r_{lr}\right) \tag{A.11}$$

shall be determined. Since $2r_{lr}$ is constant, only

$$\underset{\boldsymbol{x}_i}{\nabla} \sqrt{(\boldsymbol{x}_j - \boldsymbol{x}_i)^\mathsf{T}(\boldsymbol{x}_j - \boldsymbol{x}_i)} \tag{A.12}$$

is of interest. By applying the chain rule, first

$$\underset{\boldsymbol{x}_i}{\nabla} c_{\text{npt}} = \frac{1}{2\sqrt{(\boldsymbol{x}_j - \boldsymbol{x}_i)^\mathsf{T}(\boldsymbol{x}_j - \boldsymbol{x}_i)}} \underset{\boldsymbol{x}_i}{\nabla} \left((\boldsymbol{x}_j - \boldsymbol{x}_i)^\mathsf{T}(\boldsymbol{x}_j - \boldsymbol{x}_i)\right) \tag{A.13}$$

is obtained for the outer derivative. Next, the inner derivative is computed by first explicitly taking the dot product:

$$(\boldsymbol{x}_j - \boldsymbol{x}_i)^\mathsf{T}(\boldsymbol{x}_j - \boldsymbol{x}_i) = \sum_{k=1}^{3} x_{jk}^2 - 2x_{jk}x_{ik} + x_{ik}^2\,. \tag{A.14}$$

Applying

$$\underset{\boldsymbol{x}_i}{\nabla} = \begin{pmatrix} \frac{\partial}{\partial x_{i1}} \\ \frac{\partial}{\partial x_{i2}} \\ \frac{\partial}{\partial x_{i3}} \end{pmatrix} \tag{A.15}$$

to the inner product yields:

$$\underset{\boldsymbol{x}_i}{\nabla} \left((\boldsymbol{x}_j - \boldsymbol{x}_i)^\mathsf{T}(\boldsymbol{x}_j - \boldsymbol{x}_i)\right) = \begin{pmatrix} -2x_{j1} + 2x_{i1} \\ -2x_{j2} + 2x_{i2} \\ -2x_{j3} + 2x_{i3} \end{pmatrix} = -2\boldsymbol{x}_j + 2\boldsymbol{x}_i = -2(\boldsymbol{x}_j - \boldsymbol{x}_i)\,. \tag{A.16}$$

Thus, the overall gradient can be simplified to:

$$\nabla_{\boldsymbol{x}_i} c_{\text{npt}} = -\frac{1}{2\sqrt{(\boldsymbol{x}_j - \boldsymbol{x}_i)^\intercal (\boldsymbol{x}_j - \boldsymbol{x}_i)}} 2(\boldsymbol{x}_j - \boldsymbol{x}_i) \tag{A.17}$$

$$= -\frac{(\boldsymbol{x}_j - \boldsymbol{x}_i)}{|\boldsymbol{x}_j - \boldsymbol{x}_i|} =: -\boldsymbol{n}_{ij}\,. \tag{A.18}$$

Similarly, the gradient

$$\nabla_{\boldsymbol{x}_j} \left((\boldsymbol{x}_j - \boldsymbol{x}_i)^\intercal (\boldsymbol{x}_j - \boldsymbol{x}_i) \right) = \begin{pmatrix} 2x_{j1} - 2x_{i1} \\ 2x_{j2} - 2x_{i2} \\ 2x_{j3} - 2x_{i3} \end{pmatrix} = 2(\boldsymbol{x}_j - \boldsymbol{x}_i) \tag{A.19}$$

for j, which implies:

$$\nabla_{\boldsymbol{x}_j} c_{\text{npt}} = \frac{1}{2\sqrt{(\boldsymbol{x}_j - \boldsymbol{x}_i)^\intercal (\boldsymbol{x}_j - \boldsymbol{x}_i)}} 2(\boldsymbol{x}_j - \boldsymbol{x}_i) \tag{A.20}$$

$$= \frac{(\boldsymbol{x}_j - \boldsymbol{x}_i)}{|\boldsymbol{x}_j - \boldsymbol{x}_i|} = \boldsymbol{n}_{ij}\,. \tag{A.21}$$

A.3 | Optimal Linear Transformation

At this point, the optimization problem

$$\min \sum_i |\boldsymbol{T}_{\text{rbo}} \bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^*|^2 \tag{A.22}$$

will be examined in more detail. In the following for readability $\boldsymbol{T} := \boldsymbol{T}_{\text{rbo}}$. The term to be minimized represents a scalar function $f(\boldsymbol{T})$ dependent on a matrix $\boldsymbol{T}$. The minimum of a scalar function can be found by taking the derivative with respect to the argument and setting it to zero, therefore:

$$\frac{\partial}{\partial \boldsymbol{T}} f(\boldsymbol{T}) \stackrel{!}{=} \boldsymbol{0}\,. \tag{A.23}$$

Analogous as the derivative of a scalar function with respect to a vector (the gradient) results in a vector, the derivative of a scalar function with respect to a matrix results in a matrix of the same shape (therefore, $\boldsymbol{0} \in \mathbb{R}^{3\times 3}$). First, a single summand

$$\begin{aligned}
|\boldsymbol{T}\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^*|^2 &= (\boldsymbol{T}\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^*)^\mathsf{T}(\boldsymbol{T}\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^*) && \text{(A.24)}\\
&= (\bar{\boldsymbol{x}}_i^\mathsf{T}\boldsymbol{T}^\mathsf{T} - \bar{\boldsymbol{x}}_i^{*\mathsf{T}})(\boldsymbol{T}\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^*) && \text{(A.25)}\\
&= \bar{\boldsymbol{x}}_i^\mathsf{T}\boldsymbol{T}^\mathsf{T}\boldsymbol{T}\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^{*\mathsf{T}}\boldsymbol{T}\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^\mathsf{T}\boldsymbol{T}^\mathsf{T}\bar{\boldsymbol{x}}_i^* - \bar{\boldsymbol{x}}_i^{*\mathsf{T}}\bar{\boldsymbol{x}}_i^* && \text{(A.26)}\\
&= \bar{\boldsymbol{x}}_i^\mathsf{T}\boldsymbol{T}^\mathsf{T}\boldsymbol{T}\bar{\boldsymbol{x}}_i - 2\bar{\boldsymbol{x}}_i^{*\mathsf{T}}\boldsymbol{T}\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i^{*\mathsf{T}}\bar{\boldsymbol{x}}_i^* && \text{(A.27)}
\end{aligned}$$

is considered. For all elements of this summand, known rules for matrix differentiation exist:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{T}} \left(\bar{\boldsymbol{x}}_i^\mathsf{T}\boldsymbol{T}^\mathsf{T}\boldsymbol{T}\bar{\boldsymbol{x}}_i\right) &= 2\boldsymbol{T}\bar{\boldsymbol{x}}_i\bar{\boldsymbol{x}}_i^\mathsf{T}\,, && \text{(A.28)}\\
\frac{\partial}{\partial \boldsymbol{T}} \left(2\bar{\boldsymbol{x}}_i^{*\mathsf{T}}\boldsymbol{T}\bar{\boldsymbol{x}}_i\right) &= 2\bar{\boldsymbol{x}}_i^*\bar{\boldsymbol{x}}_i^\mathsf{T}\,, && \text{(A.29)}\\
\frac{\partial}{\partial \boldsymbol{T}} \left(\bar{\boldsymbol{x}}_i^{*\mathsf{T}}\bar{\boldsymbol{x}}_i^*\right) &= \boldsymbol{0} \in \mathbb{R}^{3\times 3}\,. && \text{(A.30)}
\end{aligned}$$

Consequently, the entire expression yields:

$$\frac{\partial}{\partial \boldsymbol{T}} f(\boldsymbol{T}) = \sum_i 2\boldsymbol{T}\bar{\boldsymbol{x}}_i\bar{\boldsymbol{x}}_i^\mathsf{T} - 2\bar{\boldsymbol{x}}_i^*\bar{\boldsymbol{x}}_i^\mathsf{T} \stackrel{!}{=} \boldsymbol{0}\,. \tag{A.31}$$

This yields:

$$\boldsymbol{T}\sum_i \bar{\boldsymbol{x}}_i\bar{\boldsymbol{x}}_i^\mathsf{T} = \sum_i \bar{\boldsymbol{x}}_i^*\bar{\boldsymbol{x}}_i^\mathsf{T}\,. \tag{A.32}$$

Overall, this results in:

$$\boldsymbol{T} = \left(\sum_i \bar{\boldsymbol{x}}_i^*\bar{\boldsymbol{x}}_i^\mathsf{T}\right)\left(\sum_i \bar{\boldsymbol{x}}_i\bar{\boldsymbol{x}}_i^\mathsf{T}\right)^{-1}\,, \tag{A.33}$$

under the condition that the matrix $\sum_i \bar{\boldsymbol{x}}_i\bar{\boldsymbol{x}}_i^\mathsf{T}$ is invertible.

Bibliography

[A20] G. Asimellis. *Wave Optics*. Vol. 3. Lectures in Optics. Spie Press, 2020. DOI: 10.1117/3.2506314.

[A68] A. Appel. "Some techniques for shading machine renderings of solids". In: *Proceedings of the Spring Joint Computer Conference (AFIPS)*. 1968, pp. 37–45. DOI: 10.1145/1468075.1468082.

[AA02] T. Akenine-Möller and U. Assarsson. "Approximate soft shadows on arbitrary surfaces using penumbra wedges". In: *Proceedings of the 13th Eurographics Workshop on Rendering*. 2002, pp. 297–306. DOI: 10.2312/egwr/egwr02/297-306.

[AA03] U. Assarsson and T. Akenine-Möller. "A geometry-based soft shadow volume algorithm using graphics hardware". In: *ACM Transactions on Graphics (TOG)* 22.3 (2003), pp. 511–520. DOI: 10.1145/882262.882300.

[AHH+18] T. Akenine-Möller et al. *Real-Time Rendering*. 4th ed. CRC Press, 2018. DOI: 10.1201/b22086.

[AHL+06] L. Atty et al. "Soft shadow maps: Efficient sampling of light source visibility". In: *Computer Graphics Forum* 25.4 (2006), pp. 725–741. DOI: 10.1111/j.1467-8659.2006.00995.x.

[AIA+12] N. Akinci et al. "Versatile rigid-fluid coupling for incompressible SPH". In: *ACM Transactions on Graphics (TOG)* 31.4, article no. 62 (2012). DOI: 10.1145/2185520.2185558.

[AJG+23] A. M. Al-Ansi et al. "Analyzing augmented reality (AR) and virtual reality (VR) recent development in education". In: *Social Sciences & Humanities Open* 8.1, article no. 100532 (2023). DOI: 10.1016/j.ssaho.2023.100532.

[AMB+07] T. Annen et al. “Convolution shadow maps”. In: *Proceedings of the 18th Eurographics Symposium on Rendering (EGSR)*. 2007, pp. 51–60. DOI: 10.2312/egwr/egsr07/051-060.

[AO11] I. Alduán and M. A. Otaduy. “SPH granular flow with friction and cohesion”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2011, pp. 25–32. DOI: 10.1145/2019406.2019410.

[ARB+03] S. Agarwal et al. “Structured importance sampling of environment maps”. In: *ACM Transactions on Graphics (TOG)* 22.3 (2003), pp. 605–612. DOI: 10.1145/882262.882314.

[ARH+00] M. Agrawala et al. “Efficient image-based methods for rendering soft shadows”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 2000, pp. 375–384. DOI: 10.1145/344779.344954.

[AT20] A. Alhakamy and M. Tuceryan. “Real-time illumination and visual coherence for photorealistic augmented/mixed reality”. In: *ACM Computing Surveys (CSUR)* 53.3, article no. 49 (2020). DOI: 10.1145/3386496.

[AT97] T. Akenine-Möller and B. Trumbore. “Fast, minimum storage ray-triangle intersection”. In: *Journal of Graphics Tools* 2.1 (1997), pp. 21–28. DOI: 10.1080/10867651.1997.10487468.

[ATO09] I. Alduan, Á. Tena, and M. A. Otaduy. “Simulation of high-resolution granular media”. In: *Proceedings of the Spanish Computer Graphics Conference (CEIG)*. 2009, pp. 11–18. DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/011-018.

[AvC18] A. Alwosheel, S. van Cranenburgh, and C. G. Chorus. “Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis”. In: *Journal of Choice Modelling* 28 (2018), pp. 167–182. DOI: 10.1016/j.jocm.2018.07.002.

[B06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[B07] R. Bridson. “Fast poisson disk sampling in arbitrary dimensions”. In: *ACM SIGGRAPH Sketches*. 2007, p. 22. DOI: 10.1145/1278780.1278807.

[B15] R. Bridson. *Fluid Simulation for Computer Graphics*. 2nd ed. A. K. Peters, 2015. DOI: 10.1201/9781315266008.

[B23] S. Bian. “Research on the application of VR in games”. In: *Highlights in Science, Engineering and Technology* 39 (2023), pp. 389–394. DOI: 10.54097/hset.v39i.6558.

[B77] J. F. Blinn. “Models of light reflection for computer synthesized pictures”. In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1977, pp. 192–198. DOI: 10.1145/563858.563893.

[BB18] H. M. Beakawi Al-Hashemi and O. S. Baghabra Al-Amoudi. “A review on the angle of repose of granular materials”. In: *Powder Technology* 330 (2018), pp. 397–417. DOI: 10.1016/j.powtec.2018.02.003.

[BB37] A. Bravais and L. Bravais. “Essai sur la disposition des feuilles curvisériées”. In: *Annales des sciences naturelles (Botanique)* 7 (1837), pp. 42–110.

[BFA02] R. Bridson, R. Fedkiw, and J. Anderson. “Robust treatment of collisions, contact and friction for cloth animation”. In: *ACM Transactions on Graphics (TOG)* 21.3 (2002), pp. 594–603. DOI: 10.1145/566654.566623.

[BH85] M. Brooks and B. Horn. “Shape and source from shading”. In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*. 1985, pp. 932–936.

[BKM+07] R. Balevičius et al. “Microscopic and macroscopic analysis of granular material behaviour in 3D flat-bottomed hopper by the discrete element method”. In: *Archives of Mechanics* 59.3 (2007), pp. 231–257.

[BKP+10] M. Botsch et al. *Polygon Mesh Processing*. 1st ed. A. K. Peters, 2010. DOI: 10.1201/b10688.

[BKR88] J. Brackbill, D. Kothe, and H. Ruppel. “Flip: A low-dissipation, particle-in-cell method for fluid flow”. In: *Computer Physics Communications* 48.1 (1988), pp. 25–38. DOI: 10.1016/0010-4655(88)90020-3.

[BL07] M. A. Brown and D. G. Lowe. “Automatic panoramic image stitching using invariant features”. In: *International Journal of Computer Vision* 74 (2007), pp. 59–73. DOI: 10.1007/s11263-006-0002-3.

[BM15] J. T. Barron and J. Malik. “Shape, illumination, and reflectance from shading”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.8 (2015), pp. 1670–1687. DOI: 10.1109/tpami.2014.2377712.

[BWW+10] J. Bowers et al. “Parallel poisson disk sampling with spectrum analysis on surfaces”. In: *ACM Transactions on Graphics (TOG)* 29.6, article no. 166 (2010). DOI: 10.1145/1882261.1866188.

[BYM05] N. Bell, Y. Yu, and P. J. Mucha. “Particle-based simulation of granular materials”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2005, pp. 77–86. DOI: 10.1145/1073368.1073379.

[C77] F. C. Crow. “Shadow algorithms for computer graphics”. In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1977, pp. 242–248. DOI: 10.1145/563858.563901.

[C86] R. L. Cook. “Stochastic sampling in computer graphics”. In: *ACM Transactions on Graphics (TOG)* 5.1 (1986), pp. 51–72. DOI: 10.1145/7529.8927.

[C87] J. Camille. “Cours d’analyse de l’eć ole polytechnique”. In: Paris: Gauthier-Villars et fils., 1887, pp. 587–594.

[CD03] E. Chan and F. Durand. “Rendering fake soft shadows with smoothies”. In: *Proceedings of the 14th Eurographics Symposium on Rendering (EGSR)*. 2003, pp. 208–218. DOI: 10.2312/egwr/egwr03/208-218.

[CS79] P. A. Cundall and O. D. L. Strack. “A discrete numerical model for granular assemblies”. In: *Géotechnique* 29.1 (1979), pp. 47–65. DOI: 10.1680/geot.1979.29.1.47.

[CS99] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices, and Groups*. 3rd ed. Springer, 1999. DOI: 10.1007/978-1-4757-6568-7.

[CSC+18] D. Cheng et al. “Learning scene illumination by pairwise photos from rear and front mobile cameras”. In: *Computer Graphics Forum* 37.7 (2018), pp. 213–221. DOI: 10.1111/cgf.13561.

[CUH16] D. Clevert, T. Unterthiner, and S. Hochreiter. “Fast and accurate deep network learning by exponential linear units (ELUs)”. In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. 2016. DOI: 10.48550/arXiv.1511.07289.

[D12] L. Deng. “The MNIST database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/msp.2012.2211477.

[D16] G. K. Das. “Sediment grain size”. In: *Encyclopedia of Estuaries*. Springer, 2016, pp. 555–558.

[D98] P. Debevec. "Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography". In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1998, pp. 189–198. DOI: 10.1145/280814.280864.

[DBK+21] A. Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *Proceedings of the 9th International Conference on Learning Representations (ICLR)*. 2021. DOI: 10.48550/arXiv.2010.11929.

[DCB16] C. Deul, P. Charrier, and J. Bender. "Position-based rigid-body dynamics". In: *Computer Animation and Virtual Worlds* 27.2 (2016), pp. 103–112. DOI: 10.1002/cav.1614.

[DD02] F. Durand and J. Dorsey. "Fast bilateral filtering for the display of high-dynamic-range images". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* 21.3 (2002), pp. 257–266. DOI: 10.1145/566570.566574.

[DDA+14] P. Desai et al. "A review paper on oculus rift–a virtual reality headset". In: *International Journal of Engineering Trends and Technology* 13.4 (2014), pp. 175–179. DOI: 10.14445/22315381/ijett-v13p237.

[DDS+09] J. Deng et al. "ImageNet: A large-scale hierarchical image database". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255. DOI: 10.1109/cvpr.2009.5206848.

[DEH+23] M. R. K. Dastjerdi et al. "EverLight: Indoor-outdoor editable HDR lighting estimation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, pp. 7420–7429. DOI: 10.1109/iccv51070.2023.00682.

[DM97] P. E. Debevec and J. Malik. "Recovering high dynamic range radiance maps from photographs". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1997, pp. 369–378. DOI: 10.1145/258734.258884.

[DMA+03] F. Drago et al. "Adaptive logarithmic mapping for displaying high contrast scenes". In: *Computer Graphics Forum* 22.3 (2003), pp. 419–426. DOI: 10.1111/1467-8659.00689.

[DOD+20] J. M. Davila Delgado et al. "A research agenda for augmented and virtual reality in architecture, engineering and construction". In: *Advanced Engineering Informatics* 45, article no. 101122 (2020). DOI: 10.1016/j.aei.2020.101122.

[DP52] D. C. Drucker and W. Prager. "Soil mechanics and plastic analysis or limit design". In: *Quarterly of Applied Mathematics* 10.2 (1952), pp. 157–165. DOI: 10.1090/qam/48291.

[EAS+09] E. Eisemann et al. "Casting shadows in real time". In: *ACM SIGGRAPH Asia Courses*. 2009, article no. 21. DOI: 10.1145/1665817.1722963.

[EGI+23] M. Eswaran et al. "Augmented reality-based guidance in product assembly and maintenance/repair perspective: A state of the art review on challenges and opportunities". In: *Expert Systems with Applications* 213, article no. 118983 (2023). DOI: 10.1016/j.eswa.2022.118983.

[EMU17] G. Eilertsen, R. K. Mantiuk, and J. Unger. "A comparative review of tone-mapping algorithms for high dynamic range video". In: *Computer Graphics Forum* 36.2 (2017), pp. 565–592. DOI: 10.1111/cgf.13148.

[EVF19] M. Eckert, J. S. Volmerg, and C. M. Friedrich. "Augmented reality in medicine: Systematic and bibliographic review". In: *JMIR mHealth and uHealth* 7.4, article no. 10967 (2019). DOI: 10.2196/10967.

[F05] R. Fernando. "Percentage-closer soft shadows". In: *ACM SIGGRAPH Sketches*. 2005, p. 35. DOI: 10.1145/1187112.1187153.

[F85] R. P. Feynman. *QED: The Strange Theory of Light and Matter*. Revised. Princeton University Press, 1985. DOI: 10.2307/j.ctt2jc8td.

[FCJ07] J. R. Frisvad, N. J. Christensen, and H. W. Jensen. "Computing the scattering properties of participating media using Lorenz-Mie theory". In: *ACM Transactions on Graphics (TOG)* 26.3, article no. 60 (2007). DOI: 10.1145/1276377.1276452.

[FDT+09] A. Ferrari et al. "A new 3D parallel SPH scheme for free surface flows". In: *Computers & Fluids* 38.6 (2009), pp. 1203–1217. DOI: 10.1016/j.compfluid.2008.11.012.

[FM17] M. Frâncu and F. Moldoveanu. "Unified simulation of rigid and flexible bodies using position based dynamics". In: *Proceedings of the 13th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS)*. 2017. DOI: 10.2312/vriphys.20171083.

[G17] D. J. Griffiths. *Introduction to Electrodynamics*. 4th ed. Cambridge University Press, 2017. DOI: 10.1017/9781108333511.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[GGG+16] D. Guarnera et al. "BRDF representation and acquisition". In: *Computer Graphics Forum* 35.2 (2016), pp. 625–650. DOI: 10.1111/cgf.12867.

[GHH01] S. Gibson, T. Howard, and R. J. Hubbold. "Flexible image-based photometric reconstruction using virtual light sources". In: *Computer Graphics Forum* 20.3 (2001), pp. 203–214. DOI: 10.1111/1467-8659.00513.

[GHS+19] M.-A. Gardner et al. "Deep parametric indoor lighting estimation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 7174–7182. DOI: 10.1109/iccv.2019.00727.

[GK24] S. Gupta and J. Keyser. "Adaptive sampling for simulating granular materials". In: *Proceedings of the Eurographics Symposium on Computer Animation (SCA) - Posters*. 2024. DOI: 10.2312/sca.20241168.

[GM77] R. A. Gingold and J. J. Monaghan. "Smoothed particle hydrodynamics: Theory and application to non-spherical stars". In: *Monthly Notices of the Royal Astronomical Society* 181.3 (1977), pp. 375–389. DOI: 10.1093/mnras/181.3.375.

[GRS12] L. Gruber, T. Richter-Trummer, and D. Schmalstieg. "Real-time photometric registration from arbitrary geometry". In: *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2012, pp. 119–128. DOI: 10.1109/ismar.2012.6402548.

[GSH+19] M. Garon et al. "Fast spatially-varying indoor lighting estimation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 6901–6910. DOI: 10.1109/cvpr.2019.00707.

[GSY+17] M.-A. Gardner et al. "Learning to predict indoor illumination from a single image". In: *ACM Transactions on Graphics (TOG)* 36.6, article no. 176 (2017). DOI: 10.1145/3130800.3130891.

[GW18] R. Gonzalez and R. Woods. *Digital Image Processing*. 4th ed. Pearson Education, 2018.

[H01] E. Haines. "Soft planar shadows using plateaus". In: *Journal of Graphics Tools* 6.1 (2001), pp. 19–27. DOI: 10.1080/10867651.2001.10487534.

[H14] D. Holz. "Parallel particles (P2): A parallel position based approach for fast and stable simulation of granular materials". In: *Proceedings of the 11th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS)*. 2014. DOI: 10.2312/vriphys.20141232.

[H17] E. Hecht. *Optics*. 5th ed. Pearson Education, 2017.

[H86] N. J. Higham. "Computing the polar decomposition—With applications". In: *SIAM Journal on Scientific and Statistical Computing* 7.4 (1986), pp. 1160–1174. DOI: 10.1137/0907079.

[HF04] S. D. Hordley and G. D. Finlayson. "Re-evaluating colour constancy algorithms". In: *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)*. 2004, pp. 76–79. DOI: 10.1109/icpr.2004.1334009.

[HFG+18] Y. Hu et al. "A moving least squares material point method with displacement discontinuity and two-way rigid body coupling". In: *ACM Transactions on Graphics (TOG)* 37.4, article no. 150 (2018). DOI: 10.1145/3197517.3201293.

[HH11] M. Hapala and V. Havran. "Review: Kd-tree traversal algorithms for ray tracing". In: *Computer Graphics Forum* 30.1 (2011), pp. 199–213. DOI: https://doi.org/10.1111/j.1467-8659.2010.01844.x.

[HH96] M. Herf and P. S. Heckbert. "Fast soft shadows". In: *ACM SIGGRAPH Visual Proceedings*. 1996, p. 145. DOI: 10.1145/253607.253870.

[HK20] D. H. House and J. C. Keyser. *Foundations of Physically Based Modeling and Animation*. 1st ed. CRC Press, 2020. DOI: 10.1201/9781315373140.

[HLA+19] Y. Hu et al. "Taichi: A language for high-performance computation on spatially sparse data structures". In: *ACM Transactions on Graphics (TOG)* 38.6, article no. 201 (2019). DOI: 10.1145/3355089.3356506.

[HLH+03] J.-M. Hasenfratz et al. "A survey of real-time soft shadows algorithms". In: *Computer Graphics Forum* 22.4 (2003), pp. 753–774. DOI: 10.1111/j.1467-8659.2003.00722.x.

[HLW17] G. Huang, Z. Liu, and K. Q. Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: 10.1109/cvpr.2017.243.

[HSH+17] Y. Hold-Geoffroy et al. “Deep outdoor illumination estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2373–2382. DOI: 10.1109/cvpr.2017.255.

[HSM+19] M. Howard et al. “Quantized bounding volume hierarchies for neighbor search in molecular simulations on graphics processing units”. In: *Computational Materials Science* 164 (2019), pp. 139–146. DOI: 10.1016/j.commatsci.2019.04.004.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[HWH19] H. Ham, J. Wesley, and H. Hendra. “Computer vision based 3D reconstruction: A review”. In: *International Journal of Electrical and Computer Engineering (IJECE)* 9.4 (2019), pp. 2394–2402. DOI: 10.11591/ijece.v9i4.pp2394-2402.

[HZ04] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. DOI: 10.1017/cbo9780511811685.

[HZR+16] K. He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778. DOI: 10.1109/cvpr.2016.90.

[I15] International Telecommunication Union. *Recommendation BT.709-6: Parameter values for the HDTV standards for production and international programme exchange*. 2015.

[IAB+11] M. Ihmsen et al. “A parallel SPH implementation on multi-core CPUs”. In: *Computer Graphics Forum* 30.1 (2011), pp. 99–112. DOI: 10.1111/j.1467-8659.2010.01832.x.

[ICG86] D. S. Immel, M. F. Cohen, and D. P. Greenberg. “A radiosity method for non-diffuse environments”. In: *ACM SIGGRAPH Computer Graphics* 20.4 (1986), pp. 133–142. DOI: 10.1145/15886.15901.

[IDB+17] F. P. Incropera et al. *Incropera’s Principles of Heat and Mass Transfer*. 8th ed. John Wiley & Sons, 2017.

[IS15] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*. 2015, pp. 448–456. DOI: 10.48550/arxiv.1502.03167.

[IWT12] M. Ihmsen, A. Wahl, and M. Teschner. “High-resolution simulation of granular material with SPH”. In: *Proceedings of the 9th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS)*. 2012, pp. 53–60. DOI: 10.2312/PE/vriphys/vriphys12/053-060.

[JB02] H. W. Jensen and J. Buhler. “A rapid hierarchical rendering technique for translucent materials”. In: *ACM Transactions on Graphics (TOG)* 21.3 (2002), pp. 576–581. DOI: 10.1145/566654.566619.

[JC16] K. Janocha and W. Czarnecki. “On loss functions for deep neural networks in classification”. In: *Schedae Informaticae* 25 (2016), pp. 49–59. DOI: 10.4467/20838476si.16.004.6185.

[JKS13] A. Jacobson, L. Kavan, and O. Sorkine-Hornung. “Robust inside-outside segmentation using generalized winding numbers”. In: *ACM Transactions on Graphics (TOG)* 32.4, article no. 33 (2013). DOI: 10.1145/2461912.2461916.

[JML+01] H. W. Jensen et al. “A practical model for subsurface light transport”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 2001, pp. 511–518. DOI: 10.1145/383259.383319.

[JST+16] C. Jiang et al. “The material point method for simulating continuum materials”. In: *ACM SIGGRAPH Courses*. 2016, article no. 24. DOI: 10.1145/2897826.2927348.

[K01] W. Kutta. “Beitrag zur näherungsweisen Integration totaler Differentialgleichungen”. In: *Zeitschrift für Mathematik und Physik* 46 (1901), pp. 435–453.

[K09] A. Krizhevsky. “Learning multiple layers of features from tiny images”. In: *Technical Report TR*. University of Toronto, 2009.

[K20] C. R. Kitchin. *Astrophysical Techniques*. 7th. CRC Press, 2020, pp. 77–80. DOI: 10.1201/9780429491139.

[K86] J. T. Kajiya. “The rendering equation”. In: *ACM SIGGRAPH Computer Graphics* 20.4 (1986), pp. 143–150. DOI: 10.1145/15886.15902.

[KB04] A. Krishnaswamy and G. V. Baranoski. "A biophysically-based spectral model of light interaction with human skin". In: *Computer Graphics Forum* 23.3 (2004), pp. 331–340. DOI: 10.1111/j.1467-8659.2004.00764.x.

[KGP+16] G. Klár et al. "Drucker-prager elastoplasticity for sand animation". In: *ACM Transactions on Graphics (TOG)* 35.4, article no. 103 (2016). DOI: 10.1145/2897824.2925906.

[KKB+23] M. Kouijzer et al. "Implementation of virtual reality in healthcare: A scoping review on the implementation process of virtual reality in various healthcare settings". In: *Implementation Science Communications* 4, article no. 67 (1 2023). DOI: 10.1186/s43058-023-00442-2.

[KKK+17] T. Krüger et al. *The Lattice Boltzmann Method: Principles and Practice*. 1st ed. Springer, 2017. DOI: 10.1007/978-3-319-44649-3.

[KL96] V. Krishnamurthy and M. Levoy. "Fitting smooth surfaces to dense polygon meshes". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1996, pp. 313–324. DOI: 10.1145/237170.237270.

[KM20] S. Kumar and R. Mehra. "Head tracking: A comprehensive review". In: *Industrial Engineering Journal* 13.2, article no. 1220 (2020). DOI: 10.26488/iej.13.2.1220.

[KP03] J. J. Koenderink and S. C. Pont. "Irradiation direction from texture". In: *Journal of the Optical Society of America* 20.10 (2003), pp. 1875–1882. DOI: 10.1364/josaa.20.001875.

[KS13] H. Kolivand and M. S. Sunar. "Survey of shadow volume algorithms in computer graphics". In: *IETE Technical Review* 30.1 (2013), pp. 38–46. DOI: 10.4103/0256-4602.107338.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*. 2012, pp. 1097–1105. DOI: 10.1145/3065386.

[KY02] M. Kanbara and N. Yokoya. "Geometric and photometric registration for real-time augmented reality". In: *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2002, pp. 279–280. DOI: 10.1109/ismar.2002.1115112.

[L02] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002. DOI: 10.1017/cbo9780511791253.

[L17] J. Lanier. *Dawn of the New Everything: Encounters with Reality and Virtual Reality*. 1st. Henry Holt and Co., 2017.

[L32] R. Likert. "A technique for the measurement of attitudes". In: *Archives of Psychology* 140 (1932), pp. 5–55.

[LBB+98] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[LD08] A. Lagae and P. Dutré. "A comparison of methods for generating Poisson disk distributions". In: *Computer Graphics Forum* 27.1 (2008), pp. 114–129. DOI: https://doi.org/10.1111/j.1467-8659.2007.01100.x.

[LD09] T. Lenaerts and P. Dutré. "Mixing fluids and granular materials". In: *Computer Graphics Forum* 28.2 (2009), pp. 213–218. DOI: 10.1111/j.1467-8659.2009.01360.x.

[LE20] L. Liang and S. Elliot. "A systematic review of augmented reality tourism research: What is now and what is next?" In: *Tourism and Hospitality Research* 21.1 (2020), pp. 15–30. DOI: 10.1177/1467358420941913.

[LFC28] H. Lewy, K. Friedrichs, and R. Courant. "Über die partiellen Differenzengleichungen der mathematischen Physik". In: *Mathematische Annalen* 100 (1928), pp. 32–74. DOI: 10.1007/bf01448839.

[LGH+13] J. Lopez-Moreno et al. "Multiple light source estimation in a single image". In: *Computer Graphics Forum* 32.8 (2013), pp. 170–182. DOI: 10.1111/cgf.12195.

[LKG+19] C. Liu et al. "PlaneRCNN: 3D plane detection and reconstruction from a single image". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4445–4454. DOI: 10.1109/cvpr.2019.00458.

[LL10] M. Liu and G. Liu. "Smoothed particle hydrodynamics (SPH): An overview and recent developments". In: *Archives of Computational Methods in Engineering* 17 (2010), pp. 25–76. DOI: 10.1007/s11831-010-9040-7.

[LMT22] V. Lavoye, J. Mero, and A. Tarkiainen. "Augmented reality in retail and e-commerce: A literature review". In: *From Micro to Macro: Dealing with Uncertainties in the Global Marketplace*. 2022, pp. 211–212. DOI: 10.1007/978-3-030-89883-0_55.

[LN12] S. Lombardi and K. Nishino. "Reflectance and natural illumination from a single image". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2012, pp. 582–595. DOI: 10.1007/978-3-642-33783-3_42.

[LN16] S. Lombardi and K. Nishino. "Reflectance and illumination recovery in the wild". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.1 (2016), pp. 129–141. DOI: 10.1109/tpami.2015.2430318.

[LW93] E. Lafortune and Y. Willems. "Bi-directional path tracing". In: *Proceedings of the 3rd International Conference on Computational Graphics and Visualization Techniques (Compugraphics)*. 1993, pp. 145–153.

[LWT+15] Y. Li et al. "A survey of recent advances in visual feature detection". In: *Neurocomputing* 149 (2015), pp. 736–751. DOI: 10.1016/j.neucom.2014.08.003.

[M08] D. Morin. *Introduction to Classical Mechanics: With Problems and Solutions*. Cambridge University Press, 2008, pp. 24–26. DOI: 10.1017/CBO9780511808951.

[M19] E. Milotti. "Mathematical models of noise". In: *The Physics of Noise*. Morgan & Claypool, 2019, pp. 1–41. DOI: 10.1088/2053-2571/ab3c46ch3.

[M92] J. J. Monaghan. "Smoothed particle hydrodynamics". In: *Annual Review of Astronomy and Astrophysics* 30 (1992), pp. 543–574. DOI: 10.1146/annurev.aa.30.090192.002551.

[MBP+22] S. Minaee et al. "Image segmentation using deep learning: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2022), pp. 3523–3542. DOI: 10.1109/tpami.2021.3059968.

[MBW+23] T. T. Minh Tran et al. "Wearable augmented reality: Research trends and future directions from three major venues". In: *IEEE Transactions on Visualization and Computer Graphics* 29.11 (2023), pp. 4782–4793. DOI: 10.1109/tvcg.2023.3320231.

[MG22] A. Marto and A. Gonçalves. "Augmented reality games and presence: A systematic review". In: *Journal of Imaging* 8.4, article no. 91 (2022). DOI: 10.3390/jimaging8040091.

[MHH+07] M. Müller et al. “Position based dynamics”. In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118. DOI: 10.1016/j.jvcir.2007.01.005.

[MHT+05] M. Müller et al. “Meshless deformations based on shape matching”. In: *ACM Transactions on Graphics (TOG)* 24.3 (2005), pp. 471–478. DOI: 10.1145/1073204.1073216.

[MKR+11] R. Mantiuk et al. “HDR-VDP-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions”. In: *ACM Transactions on Graphics (TOG)* 30.4, article no. 40 (2011). DOI: 10.1145/2010324.1964935.

[MMC+14] M. Macklin et al. “Unified particle physics for real-time applications”. In: *ACM Transactions on Graphics (TOG)* 33.4, article no. 153 (2014). DOI: 10.1145/2601097.2601152.

[MMC+20] M. Müller et al. “Detailed rigid body simulation with extended position based dynamics”. In: *Computer Graphics Forum* 39.8 (2020), pp. 101–112. DOI: 10.1111/cgf.14105.

[MMC16] M. Macklin, M. Müller, and N. Chentanez. “XPBD: Position-based simulation of compliant constrained dynamics”. In: *Proceedings of the 9th International Conference on Motion in Games (MIG)*. 2016, pp. 49–54. DOI: 10.1145/2994258.2994272.

[MMP87] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. “The discrete geodesic problem”. In: *SIAM Journal of Computing* 16.4 (1987), pp. 647–668. DOI: 10.1137/0216045.

[MOB+21] D. Meister et al. “A survey on bounding volume hierarchies for ray tracing”. In: *Computer Graphics Forum* 40.2 (2021), pp. 683–712. DOI: https://doi.org/10.1111/cgf.142662.

[MST+21] B. Mildenhall et al. “NeRF: Representing scenes as neural radiance fields for view synthesis”. In: *Communications of the ACM* 65.1 (2021), pp. 99–106. DOI: 10.1145/3503250.

[MTK+23] A. Marougkas et al. “Virtual reality in education: A review of learning theories, approaches and methodologies for the last decade”. In: *Electronics* 12.13, article no. 2832 (2023). DOI: 10.3390/electronics12132832.

[MTU+95] P. Milgram et al. “Augmented reality: A class of displays on the reality-virtuality continuum”. In: *Telemanipulator and Telepresence Technologies*. 1995, pp. 282–292. DOI: 10.1117/12.197321.

[MZ88] G. R. McNamara and G. Zanetti. “Use of the Boltzmann equation to simulate lattice-gas automata”. In: *Physical Review Letters* 61.20 (1988), pp. 2332–2335. DOI: 10.1103/physrevlett.61.2332.

[N65] F. E. Nicodemus. “Directional reflectance and emissivity of an opaque surface”. In: *Applied Optics* 4.7 (1965), pp. 767–775. DOI: 10.1364/ao.4.000767.

[NAA+23] S. Naeem et al. “An unsupervised machine learning algorithms: Comprehensive review”. In: *International Journal of Computing and Digital Systems* 13.1 (2023), pp. 911–921. DOI: 10.12785/ijcds/130172.

[NAR18] J. R. Nuñez, C. R. Anderton, and R. S. Renslow. “Optimizing colormaps with consideration for color vision deficiency to enable accurate interpretation of scientific data”. In: *Plos One* 13.7, article no. 199239 (2018). DOI: 10.1371/journal.pone.0199239.

[NDN96] T. Nishita, Y. Dobashi, and E. Nakamae. “Display of clouds taking into account multiple anisotropic scattering and sky light”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1996, pp. 379–386. DOI: 10.1145/237170.237277.

[NE01] P. Nillius and J.-O. Eklundh. “Automatic estimation of the projected light source direction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2001, pp. 1076–1083. DOI: 10.1109/cvpr.2001.990650.

[NGL10] R. Narain, A. Golas, and M. C. Lin. “Free-flowing granular materials with two-way solid coupling”. In: *ACM Transactions on Graphics (TOG)* 29.6, article no. 173 (2010). DOI: 10.1145/1882261.1866195.

[NOB+16] B. Nuernberger et al. “SnapToReality: Aligning augmented reality to the real world”. In: *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*. 2016, pp. 1233–1244. DOI: 10.1145/2858036.2858250.

[NRH+77] F. E. Nicodemus et al. *Geometrical Considerations and Nomenclature for Reflectance*. National Institute of Standards and Technology, 1977. DOI: 10.6028/nbs.mono.160.

[P10] V. L. Popov. "Coulomb's law of friction". In: *Contact Mechanics and Friction: Physical Principles and Applications*. Springer, 2010, pp. 133–154. DOI: 10.1007/978-3-642-10803-7_10.

[P75] B. T. Phong. "Illumination for computer generated pictures". In: *Communications of the ACM* 18.6 (1975), pp. 311–317. DOI: 10.1145/360825.360839.

[P82] A. P. Pentland. "Finding the illuminant direction". In: *Journal of the Optical Society of America* 72.4 (1982), pp. 448–455. DOI: 10.1364/josa.72.000448.

[PER+18] R. Palmarini et al. "A systematic review of augmented reality applications in maintenance". In: *Robotics and Computer-Integrated Manufacturing* 49 (2018), pp. 215–228. DOI: 10.1016/j.rcim.2017.06.002.

[PJH23] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. 4th. The MIT Press, 2023.

[PSP09] A. Panagopoulos, D. Samaras, and N. Paragios. "Robust shadow and illumination estimation using a mixture model". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 651–658. DOI: 10.1109/cvpr.2009.5206665.

[PTV+07] W. H. Press et al. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. Cambridge University Press, 2007.

[R67] C. H. Reinsch. "Smoothing by spline functions". In: *Numerische Mathematik* 10 (1967), pp. 177–183. DOI: 10.1007/bf02162161.

[R81] R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. 1st. John Wiley & Sons, 1981. DOI: 10.1002/9780470316511.

[RBA+19] N. Rahaman et al. "On the spectral bias of neural networks". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019, pp. 5301–5310. DOI: 10.48550/arXiv.1806.08734.

[RBG01] J. Rolland, Y. Baillot, and A. Goon. "A survey of tracking technology for virtual environments". In: *Fundamentals of Wearable Computers and Augmented Reality*. 1st ed. CRC Press, 2001, pp. 83–128. DOI: 10.1201/9780585383590-9.

[RGJ+20] G. Rainer et al. "Unified neural encoding of BTFs". In: *Computer Graphics Forum* 39.2 (2020), pp. 167–178. DOI: 10.1111/cgf.13921.

[RJ07] A. R. Rivers and D. L. James. "FastLSM: Fast lattice shape matching for robust real-time deformation". In: *ACM Transactions on Graphics (TOG)* 26.3 (2007), pp. 82–88. DOI: 10.1145/1276377.1276480.

[RJG+19] G. Rainer et al. "Neural BTF compression and interpolation". In: *Computer Graphics Forum* 38.2 (2019), pp. 235–244. DOI: 10.1111/cgf.13633.

[RSL+24] U. Rajapaksha et al. "Deep learning-based depth estimation methods from monocular image and videos: A comprehensive survey". In: *ACM Computing Surveys* 56.12, article no. 315 (2024). DOI: 10.1145/3677327.

[RSS+02] E. Reinhard et al. "Photographic tone reproduction for digital images". In: *ACM Transactions on Graphics (TOG)* 21.3 (2002), pp. 267–276. DOI: 10.1145/566654.566575.

[RTJ95] E. Reinhard, L. Tijssen, and F. Jansen. "Environment mapping for efficient sampling of the diffuse interreflection". In: *Photorealistic Rendering Techniques*. 1995, pp. 410–422. DOI: 10.1007/978-3-642-87825-1_30.

[RZ24] A. S. A. Rabby and C. Zhang. "BeyondPixels: A comprehensive review of the evolution of neural radiance fields". Version 3. In: *arXiv e-prints* (2024). URL: https://arxiv.org/abs/2306.03000.

[S07] K. O. Stanley. "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic Programming and Evolvable Machines* 8.2 (2007), pp. 131–162. DOI: 10.1007/s10710-007-9028-8.

[S46] I. J. Schoenberg. "Contributions to the problem of approximation of equidistant data by analytic functions". In: *Quarterly of Applied Mathematics* 4 (1946), pp. 45–99. DOI: 10.1007/978-1-4899-0433-1_1.

[S62] M. Shimrat. "Algorithm 112: Position of point relative to polygon". In: *Communications of the ACM* 5.8 (1962), p. 434. DOI: 10.1145/368637.368653.

[S65] I. E. Sutherland. "The ultimate display". In: *Proceedings of the Congress of the Internation Federation of Information Processing (IFIP)*. 1965, pp. 506–508.

[S87] J. P. Snyder. *Map Projections: A Working Manual*. U.S. Government Printing Office, 1987. DOI: 10.3133/pp1395.

[SC23] T. Stuyck and H.-y. Chen. "DiffXPBD: Differentiable position-based simulation of compliant constraint dynamics". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 6.3, article no. 51 (2023). DOI: 10.1145/3606923.

[SCS94] D. Sulsky, Z. Chen, and H. Schreyer. "A particle method for history-dependent materials". In: *Computer Methods in Applied Mechanics and Engineering* 118.1 (1994), pp. 179–196. DOI: 10.1016/0045-7825(94)90112-0.

[SF19] S. Song and T. Funkhouser. "Neural illumination: Lighting prediction for indoor environments". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 6911–6919. DOI: 10.1109/cvpr.2019.00708.

[SJ06] R. Swinbank and R. James Purser. "Fibonacci grids: A novel approach to global modelling". In: *Quarterly Journal of the Royal Meteorological Society* 132.619 (2006), pp. 1769–1793. DOI: 10.1256/qj.05.227.

[SK21] G. Somanath and D. Kurz. "HDR environment map estimation for real-time augmented reality". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 11293–11301. DOI: 10.1109/cvpr46437.2021.01114.

[SKS02] P.-P. Sloan, J. Kautz, and J. Snyder. "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 2002, pp. 527–536. DOI: 10.1145/566570.566612.

[SLJ+15] C. Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. DOI: 10.1109/cvpr.2015.7298594.

[SMT+20] P. Srinivasan et al. "Lighthouse: Predicting lighting volumes for spatially-coherent illumination". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8077–8086. DOI: 10.1109/cvpr42600.2020.00810.

[SP09] B. Solenthaler and R. Pajarola. "Predictive-corrective incompressible SPH". In: *ACM Transactions on Graphics (TOG)* 28.3, article no. 40 (2009). DOI: 10.1145/1531326.1531346.

[SPG10] T. K. Sharpless, B. Postle, and D. M. German. "Pannini: A new projection for rendering wide angle perspective images". In: *Proceedings of the 6th International Conference on Computational Aesthetics in Graphics, Visualization and Imaging*. 2010, pp. 9–16.

[SS21] A. Sommer and U. Schwanecke. “LEAVEN - Lightweight surface and volume mesh sampling application for particle-based simulations”. In: *Computer Science Research Notes* 31.1 (2021), pp. 155–160. DOI: 10.24132/csrn.2021.3101.17.

[SS97] R. Szeliski and H.-Y. Shum. “Creating full view panoramic image mosaics and environment maps”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1997, pp. 251–258. DOI: 10.1145/258734.258861.

[SS98] C. Soler and F. X. Sillion. “Fast calculation of soft shadow textures using convolution”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1998, pp. 321–332. DOI: 10.1145/280814.280927.

[SSK+05] V. Surazhsky et al. “Fast exact and approximate geodesics on meshes”. In: *ACM Transactions on Graphics (TOG)* 24.3 (2005), pp. 553–560. DOI: 10.1145/1073204.1073228.

[SSS22] A. Sommer, U. Schwanecke, and E. Schömer. “Interactive high-resolution simulation of granular material”. In: *Journal of WSCG* 30 (2022), pp. 9–15. DOI: 10.24132/jwscg.2022.2.

[SSS23] A. Sommer, U. Schwanecke, and E. Schömer. “Real-time light estimation and neural soft shadows for AR indoor scenarios”. In: *Journal of WSCG* 31 (2023), pp. 71–79. DOI: 10.24132/jwscg.2023.8.

[SVI+16] C. Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826. DOI: 10.1109/cvpr.2016.308.

[SW15] N. Smith and S. van der Walt. “A better default colormap for matplotlib”. SciPy. 2015. URL: https://pyvideo.org/scipy-2015/a-better-default-colormap-for-matplotlib.html.

[SWP11] D. Scherzer, M. Wimmer, and W. Purgathofer. “A survey of real-time hard shadow mapping methods”. In: *Computer Graphics Forum* 30.1 (2011), pp. 169–186. DOI: 10.1111/j.1467-8659.2010.01841.x.

[SZ15] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. 2015, pp. 1–14. DOI: 10.48550/arXiv.1409.1556.

[THM+03] M. Teschner et al. “Optimized spatial hashing for collision detection of deformable objects”. In: *Proceedings of the Vision, Modeling, and Visualization Conference (VMV)*. 2003, pp. 47–54.

[TL94] G. Turk and M. Levoy. “Zippered polygon meshes from range images”. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1994, pp. 311–318. DOI: 10.1145/192161.192241.

[TM08] T. Tuytelaars and K. Mikolajczyk. “Local invariant feature detectors: A survey”. In: *Foundations and Trends in Computer Graphics and Vision* 3.3 (2008), pp. 177–280. DOI: 10.1561/0600000017.

[TS06] Y.-T. Tsai and Z.-C. Shih. “All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation”. In: *ACM Transactions on Graphics (TOG)* 25.3 (2006), pp. 967–976. DOI: 10.1145/1141911.1141981.

[TSM+20] M. Tancik et al. “Fourier features let networks learn high frequency functions in low dimensional domains”. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS)*. 2020, article no. 632. DOI: 10.48550/arxiv.2006.10739.

[TSS+15] H. Tjaden et al. “High-speed and robust monocular tracking”. In: *Proceedings of the 10th International Conference on Computer Vision Theory and Applications (VISAPP)*. 2015, pp. 462–471. DOI: 10.5220/0005267104620471.

[TSS16] H. Tjaden, U. Schwanecke, and E. Schömer. “Real-time monocular segmentation and pose tracking of multiple objects”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016, pp. 423–438. DOI: 10.1007/978-3-319-46493-0_26.

[TSS17] H. Tjaden, U. Schwanecke, and E. Schömer. “Real-time monocular pose estimation of 3D objects using temporally consistent local color histograms”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 124–132. DOI: 10.1109/iccv.2017.23.

[V67] L. Verlet. “Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules”. In: *Physical Review* 159.1 (1967), pp. 98–103. DOI: 10.1103/physrev.159.98.

[V79] H. Vogel. "A better way to construct the sunflower head". In: *Mathematical Biosciences* 44 (1979), pp. 179–189. DOI: 10.1016/0025-5564(79)90080-4.

[VS83] A. Van Oosterom and J. Strackee. "The solid angle of a plane triangle". In: *IEEE Transactions on Biomedical Engineering* 30.2 (1983), pp. 125–126. DOI: 10.1109/tbme.1983.325207.

[VSP+17] A. Vaswani et al. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*. 2017, pp. 6000–6010. DOI: 10.48550/arxiv.1706.03762.

[VZ04] M. Varma and A. Zisserman. "Estimating illumination direction from textured images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2004, pp. 179–186. DOI: 10.1109/cvpr.2004.1315030.

[W21] M. Wooldridge. *A Brief History of Artificial Intelligence: What It Is, Where We Are, and Where We Are Going*. Macmillan, 2021.

[W22] C. K. Wentworth. "A scale of grade and class terms for clastic sediments". In: *The Journal of Geology* 30.5 (1922), pp. 377–392. DOI: 10.1086/622910.

[W24] S. Wright. "Image blending". In: *Digital Compositing for Film and Video*. 5th ed. Routledge, 2024, pp. 149–178. DOI: 10.4324/9781032418070.

[W78] L. Williams. "Casting curved shadows on curved surfaces". In: *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1978, pp. 270–274. DOI: 10.1145/965139.807402.

[W79] T. Whitted. "An improved illumination model for shaded display". In: *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1979, p. 14. DOI: 10.1145/965103.807419.

[WBS+04] Z. Wang et al. "Image quality assessment: From error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/tip.2003.819861.

[WBW13] S. Woop, C. Benthin, and I. Wald. "Watertight ray/triangle intersection". In: *Journal of Computer Graphics Techniques (JCGT)* 2.1 (2013), pp. 65–82.

[WRG+09] J. Wang et al. "All-frequency rendering of dynamic, spatially-varying reflectance". In: *ACM Transactions on Graphics (TOG)* 28.5 (2009), pp. 1–10. DOI: 10.1145/1618452.1618479.

[WSC+95] K.-Y. Whang et al. “Octree-R: An adaptive octree for efficient ray tracing”. In: *IEEE Transactions on Visualization and Computer Graphics* 1.4 (1995), pp. 343–349. DOI: 10.1109/2945.485621.

[WWH+22] C.-Y. Wu et al. “Toward practical monocular indoor depth estimation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 3804–3814. DOI: 10.1109/cvpr52688.2022.00379.

[WYL+22] G. Wang et al. “StyleLight: HDR panorama generation for lighting estimation and editing”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2022, pp. 477–492. DOI: 10.1007/978-3-031-19784-0_28.

[XCX+24] S. Xu et al. “Local feature matching using deep learning: A survey”. In: *Information Fusion* 107, article no. 102344 (2024). DOI: 10.1016/j.inffus.2024.102344.

[XGY+22] Y. Xie et al. “PlanarRecon: Realtime 3D plane detection and reconstruction from posed monocular videos”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 6209–6218. DOI: 10.1109/cvpr52688.2022.00612.

[YKH+24] L. Yang et al. “Depth anything: Unleashing the power of large-scale unlabeled data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2024), pp. 10371–10381. DOI: 10.1109/cvpr52733.2024.00987.

[YNE23] E. Yigitbas, A. Nowosad, and G. Engels. “Supporting construction and architectural visualization through BIM and AR/VR: A systematic literature review”. In: *Proceedings of the 19th International Conference on Human-Computer Interaction (INTERACT)*. 2023, pp. 145–166. DOI: 10.1007/978-3-031-42283-6_8.

[Z00] Z. Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[ZB05] Y. Zhu and R. Bridson. “Animating sand as a fluid”. In: *ACM Transactions on Graphics (TOG)* 24.3 (2005), pp. 965–972. DOI: 10.1145/1073204.1073298.

[ZCZ+19] J. Zhao et al. “Physically based modeling and animation of landslides with MPM”. In: *The Visual Computer* 35.9 (2019), pp. 1223–1235. DOI: 10.1007/s00371-019-01709-3.

[ZPK18] Q.-Y. Zhou, J. Park, and V. Koltun. *Open3D: A Modern Library for 3D Data Processing*. 2018. DOI: 10.48550/arxiv.1801.09847.

[ZSX+06] F. Zhang et al. “Parallel-split shadow maps for large-scale virtual environments”. In: *Proceedings of the ACM International Conference on Virtual Reality Continuum and Its Applications (VRCIA)*. 2006, pp. 311–318. DOI: 10.1145/1128923.1128975.

[ZYZ+22] F. Zhan et al. “GMLight: Lighting estimation via geometric distribution approximation”. In: *IEEE Transactions on Image Processing* 31 (2022), pp. 2268–2278. DOI: 10.1109/tip.2022.3151997.

[ZZY+21] F. Zhan et al. “EMLight: Lighting estimation via spherical distribution approximation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, pp. 3287–3295. DOI: 10.1609/aaai.v35i4.16440.

Lebenslauf

[REMOVED]