Procedural Modeling and Image Synthesis for Virtual Surface Inspection Planning

Thesis approved by
the Department of Computer Science
University of Kaiserslautern-Landau
for the award of the Doctoral Degree
Doctor of Engineering (Dr.-Ing.)

to

Lovro Bosnar

Date of Defence: 29.04.2024.
Dean: Prof. Dr. Christoph Garth
Reviewer: Prof. Dr. Hans Hagen
Reviewer: Prof. Dr. Thomas Wischgoll
Reviewer: Prof. Dr. Holly Rushmeier

DE-386

A good picture is equivalent to a good deed.

– *Vincent van Gogh*

Acknowledgements

As I was finishing my master studies I was at crossroads with signs saying "industry" or "science". During that time, I came across a presentation by Dr. Petra Gospodnetić, then a PhD at the University of Kaiserslautern and Fraunhofer ITWM, about how science is very much connected to industry and how much it is needed to tackle common industry problems such as inspection of an aeroplane turbine. What I heard sounded like an answer to my search and after seeing complex geometrical structures and textures that were investigated at Fraunhofer ITWM (and how computer graphics can be applied to those!) I decided on an internship. Therefore, I would like to thank Dr. Petra Gospodnetić for making my internship possible since it led to my PhD at the University of Kaiserslautern.

First and foremost I would like to thank my supervisors Dr. Petra Gospodnetić and Markus Rauhut from Fraunhofer ITWM and Dr. Hans Hagen from the University of Kaiserslautern. I am grateful to my supervisors for ensuring that I have everything I need to perform the research and development provided in this thesis, for being there for me during my whole PhD, providing me with directions and discussions, and for making opportunities for meetings, networking and collaboration possible. I am truly grateful to Dr. Petra Gospodnetić for countless discussions and meetings, guidance and structure, pointing me to opportunities and people who made my research even richer, with this input I was always pushing my limits and becoming a better researcher and engineer. I simply do not have words for the amount of time, genuine care and effort she put into supervising me and ensuring that I had everything I needed, and for that I am grateful. I would also like to thank my supervisors at AppleseedHQ during Google Summer of Code where I learned a lot regarding rendering and materials in computer graphics.

During my PhD time, I was surrounded by people from the Image Processing department at Fraunhofer ITWM and the University of Kaiserslautern - a warm and kind environment which I hope to have in my future workplace. I am grateful to each and everyone from this environment. I am grateful that I had a chance to work with bright and kind students such as Natascha Jeziorski, Juraj Fulir, Maurice Didion, Lala Shakti Swarup Ray, Josiah Abah, Siddartha Dutta and Doria Šarić. I would like to thank Olena

Buchbinder and Mady Gruys for being there for me, full of support and care when solving administrative requirements and tasks. I am grateful to Martin Braun for all the help and time he provided for solving any IT problems. I would like to thank Dr. Claudia Redenbach and Dr. Katja Schladitz for all the discussions bridging modeling in computer graphics and mathematics. I would like to thank Dr. Tin Barišin for all the conversations and running sessions that made COVID time and hard PhD times easier. I would further like to thank Alexander Geng, Alex Keilmann, Ahmed Alshembari and Falco Hirschenberger for all the running sessions which I must say were crucial for a healthy PhD. I would like to thank all PhD and HiWi students for all the hikes, game nights, mensa time, conversations and fun. Particularly, I am grateful for all the great times with Alexander Geng, Alex Keilmann, Juraj Fulir, Damjan Hatić, Lars Nieradzik, Dasha Dobrovolskij and Jianming Yi. I would like to thank Franz Schreiber, with whom I was sharing an office, for all the laughs, good music, and great atmosphere making my PhD days spent in the office filled with positive and good vibes.

I am grateful for the fruitful collaboration with Dr. Holly Rushmeier and Donovan Kreul from Yale University.

I am happy and grateful that I had a chance to meet Luís Miguel Bastos Barrancos Fernandes and for all the amazing computer graphics rants.

In each chapter of my life, my friends were amazing support, always there for conversations and great times and thus I would like to thank Ivan Krauze, Luka Macan, Damir Sadiković, Ivan Zvonimir Kos, Borna Bešić, Leo Obadić, Robert Milijaš, Luce Ivković.

I am truly grateful for my family who was always supportive and there for me when needed - before my PhD and during my PhD. I would like to thank my parents, mother Sanja and father Damir for all the support and conversations and for teaching me to trust myself in whatever I decide to pursue. They were always there for me in every chapter of my life and for that I am grateful. I would like to thank my grandmother Katarina for all the conversations, wise words and support. I would like to thank my sister Tihana and brother Mihovil - kind, smart and amazing I am grateful to have them - for all the conversations, fun and time together which helped me a lot during my PhD. I would also like to thank Mihovil for reviewing one of my papers! I would like to thank my godfather Nikola Šantorić for all the game and film nights that made my time on PhD much nicer and relaxed.

I would like to thank my very special person Ariana Prpić who gave me an amazing amount of support in my last PhD year and for all our conversations and time together. I am truly grateful for such a kind, caring, wise and supportive person with whom I can dream and make those dreams come true.

Finally, I would thank all the people whose influence is always there in one way or another. I would like to thank my teachers from primary school, high school and university, particularly Ivana Crnjac, Mirjana Hrašovec and

Mirjana Domazet-Lošo who taught me how to think, learn, work and trust myself.

Abstract

Product manufacturing is performed in a massively automated and increasingly customized manner. However, overall production speed is limited by automation of inspection since each product has to ensure the required quality. A widespread and often-used quality assurance method is visual surface inspection. Automated surface inspection relies on an inspection plan and defect recognition algorithms. Both inspection planning and defect recognition algorithms development heavily rely on the availability of representative image data containing various product surface textures and imperfections showing a wide variety of possible surface responses to different viewing and lighting conditions. Due to the advancements in manufacturing, defects in products occur rarely, with different frequencies of appearance, followed by a subjective and laborious annotation process. Further, since the surface texture is often not relevant to product performance and thus not controlled, products with different surface textures are not treated as different product samples and thus not provided.

Motivated by aforementioned problems, this work introduces the following contributions: (1) image synthesis requirements for industrial quality inspection and a novel realistic image synthesis pipeline satisfying those requirements (Chapter 4), (2) texture synthesis requirements for industrial quality inspection and a procedural approach to parameterized surface texture modeling incorporating domain knowledge (Chapter 5) and (3) defect synthesis requirements for industrial quality inspection as well as a procedural approach to parameterized defect modeling (Chapter 6). The contributions presented in this thesis, make it possible to obtain, in a controllable and automated manner, the required amount of image data, containing realistic and varying surface textures resembling machining surfaces as well as diversified geometrical defects with automated, pixel-precise annotations (Chapters 7, 8). The presented contributions enable the inspection planning and development of machine vision algorithms for defect recognition to be performed completely virtually, by inspection planning experts, without computer graphics knowledge.

Acronyms

RGB - red, green and blue triplet
IOR - index of refraction
SSS - sub-surface scattering
BSDF - bidirectional scattering distribution function
BRDF - bidirectional reflectance distribution function
BTDF - bidirectional transmission distribution function
NDF - normal distribution function, i.e., distribution of microfacet normals
DOF - depth of field
SPD - spectral power distributuion
R - requirement
PRN - pseudo-random number
BVH - bounding volume hierarchy
BSP tree - binary space partitioning tree
LTE - light transport equation
CSG - constructive solid geometry

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Image Synthesis for Virtual Surface Inspection Planning

Image synthesis using computer graphics relies on 3D scene modeling and simulation as well as rendering which are completely controllable. Therefore, generating synthetic images using computer graphics is desirable and applicable in various disciplines. Many computer graphics applications such as games, animated films, visual effects, predictive simulations, robotics, etc. brought many contributions ranging from 3D scene modeling and simulation techniques to photo-realistic and expressive rendering algorithms making this area more and more exciting and more involved in solving problems in different disciplines.

In this work, computer graphics methods are applied for image synthesis in virtual surface inspection planning. More specifically, the real inspection environment is modeled as a 3D scene where the focus is on procedural modeling of the product surface; both textures and imperfections. Although the focus is on virtual planning which includes configuration and placement of acquisition hardware for obtaining the required product coverage as well as synthetic data generation for the development of defect recognition algorithms, the introduced computer graphics concepts can be extended and used in other disciplines where realistic and parameterized image synthesis is required.

1.1 Virtual Surface Inspection Planning

Every manufactured object, further referred to as *product*, must assure certain quality standards. *Visual surface inspection* is an often and widely used quality assurance method as discussed by Gospodnetic [3], Mohammadikaji [4] and Beyerer et al. [5]. Traditionally, visual surface inspection is performed by human experts relying on their perception. The advantage of inspection performed by a human expert is their experience, adaptation

to new scenarios and intuition. Unfortunately, this also implies a slow, inconsistent, laborious and subjective inspection process. On the other hand, automated inspection relies on cameras which are highly available and applicable to the inspection of both product shape and its surface. Acquired images are then automatically analyzed using image processing algorithms for defect recognition.

Automated visual inspection using computer vision has been widely researched and applied to achieve different quality requirements such as product dimensions, surface conditions, texture quality, optical properties as well as aesthetic and technical defect recognition (Beyerer et al. [5]). In this work, the focus is on surface inspection systems applied for defect recognition.

In the case of known inspection scenarios with known requirements, automated inspection systems are typically readily available. Unfortunately, such systems are highly specialized and hard to apply to different set of requirements. Therefore, for the majority of manufactured products, a custom automated inspection must be developed which requires planning. Inspection planning and the challenges it brings, discussed in the next section, are crucial for the work presented in this thesis since they determine the aspects on which this work focuses.

1.1.1 Inspection Planning and Challenges

Production lines are constantly adopting the concepts of smart factories and smart manufacturing introduced in *Industry 4.0* which results in improved automation and customization capabilities. However, in order to increase the efficiency of overall production, the efficiency of automated inspection must be improved. Since the availability and applicability of visual inspection make it a widely used quality assurance method, surface inspection can be considered an important part of quality assurance for Industry 4.0. Further, the complex nature of automated visual inspection development makes inspection systems very application-specific and rigid. Therefore, the flexibility required by Industry 4.0 to keep up with customized products is very difficult to achieve.

Custom automated inspection system requires *planning*. Inspection planning is a process where experts choose acquisition hardware (e.g., camera and light) and configure their spatial placement to obtain required product coverage. Further, algorithms are designed for defect recognition of acquired product images. This process is highly challenging due to several reasons. First, obtaining the required amount of products with diversified defects is challenging due to ever-increasing manufacturing capabilities or small production batches. Second, due to the complexity of product geometry, material and defects which are to be detected, inspection planning must consider the following aspects: *product shape*, *product reflectance and texture*, *defect appearance*, *light* and *camera*.

Figure 1.1: Inspected parts with the same geometry but different reflectance and texture appear significantly different.

Product shape. Product surfaces have a wide range of geometrical features such as holes, openings, curved areas, free-form regions, etc. Due to those geometrical features multiple images taken with different camera and light positions are required. Therefore, inspection planning must cover the product surface to the required extent and as efficiently as possible. Increased geometrical complexity increases the complexity of inspection which, at one point, becomes extremely challenging if performed manually.

Product reflectance and texture. To ensure required product surface coverage it is not only enough to rely on product geometry. This is because different products have different surfaces which are exhibiting different reflection and texture properties. For this reason, different surfaces will not appear the same if acquired under the same conditions. Furthermore, even products with the same geometries and the same acquisition setup but with different reflectance and texture properties will not appear the same which can be seen in Figure 1.1. That is, the amount of visible surface will be different. The type of surface material defines optical properties (i.e., reflectivity) and interaction with light, influencing the appearance of the acquired image. Furthermore, processing of product surface (e.g., machining) introduces specific texture which further influences optical properties, that is, interaction with light. The importance of reflectance and texture are crucial for this thesis and are tackled later.

Defect appearance. Defect is application specific. Its occurrence is random and unique to the production environment. Thus, a general classification of surface defects is a challenging task. Generally, it represents any kind of surface part which deviates from the expected surface. For automated visual inspection, the producer usually predefined the deviation from the expected surface. Two main points arise when it comes to defect recognition: defect characterization and defect visibility. Defects can occur in a wide range of shapes and sizes as well as on different parts of inspected objects, surrounded by a wide range of possible textures (Fig-

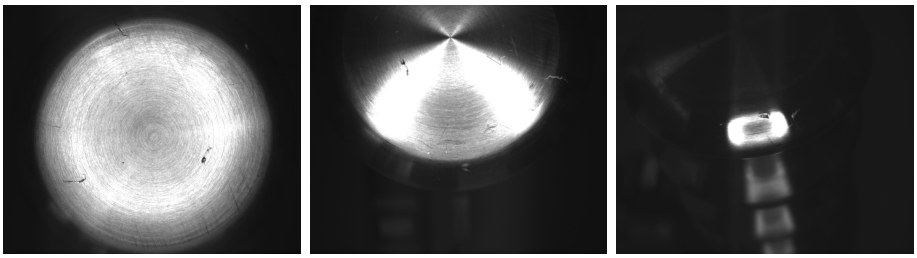


Figure 1.2: Three different viewpoints of the same inspected object. Defects with a wide range of shapes and sizes can occur on different parts of inspected objects.

ure 1.2). Therefore, obtaining the required amount of diversified defects is crucial for solving the problem of defect recognition. Defect shape, size, positioning and appearance pose crucial importance for this thesis and are tackled later.

Light. Choice of light and its position highly determines the amount of visible surface and defects which can range from completely visible to completely hidden (Figure 1.2). Based on product surface properties and types of defects, either diffuse or direct light can be used. Diffuse light illuminates the scene uniformly from all directions, thus, reducing highlights and shadows. On the other hand, direct light illuminates the surface predominantly or completely from one direction. Light can be configured so that the surface of an object appears bright while defects appear dark (i.e., bright-field). On the other hand, light can be configured so that the surface appears dark while defects appear bright since they directly reflect into the camera (i.e., dark-field).

Camera. Image analysis can be performed on images obtained using arbitrary camera systems (infrared, velocity, depth, etc.). In this work, the focus is on an industrial camera which captures the visible light spectrum (Figure 1.2). Captured spectrum can be saved as single-channel monochrome (grayscale) or multi-channel red, green and blue (RGB). The grayscale camera is frequently used, when color is not important, benefiting from higher per-pixel resolution and less data for transmission, storage and analysis. Cameras can further be area-scan or line-scan. Area-scan cameras have a fixed resolution corresponding to the camera sensor. On the other hand, a line-scan camera has a fixed resolution only along one image axis enabling to capture of large, high-resolution images capture of moving objects. An important camera parameter that has to be set is resolution determining the trade-off between image quality and analysis time. Next, the camera's field of view must be set to determine the portion of the visible product surface. Finally, based on the camera's field of view and its distance from the product surface, an appropriate lens introducing depth of field is selected. Depth of

field determines the distance between the nearest and furthest points which are in sharp focus. A larger camera aperture implies a smaller depth of field and vice versa. A larger camera aperture allows more light to reach the camera sensor which requires a higher camera shutter speed to eliminate the motion blur effect for moving objects. For curved product surfaces, a small depth of field represents a problem since only certain parts of the surface will be in focus.

1.1.2 Virtual Inspection Planning

Motivated by the discussed challenges, Gospodnetic [3], [6] introduced virtual inspection planning - migration of the physical planning process into a virtual environment. The benefits of such an environment are manyfold. First, it should provide planning experts with a set of tools which enable them to plan the inspection for products of arbitrary geometry, reflectance and texture properties. Second, the virtual environment aims to reduce the cost and time for inspection system development by enabling early product coverage and precision estimates. Finally, the virtual environment gives a way to perform planning offline and apply the obtained results on an online system and as such enables the flexibility required for Industry 4.0.

Virtual inspection planning is based on a virtualization tool - an interactive inspection planning tool named V-POI [7], [6]. This tool aims to replicate the physical inspection planning process in a virtual environment. Specifically, it enables the use of CAD models of inspected products as well as real-world camera properties for planning. This way, acquisition hardware (camera and light) configuration and placement can be done completely virtually with precise information on geometrical coverage. Hence, viewpoint placement evaluation is estimated by geometrical product surface coverage given different camera positions [8].

The introduced virtual inspection planning environment and its goals represent the foundation for this thesis. Several critical research and development directions are identified which extend the virtual planning platform and thus are crucial for inspection planning and automation purposes.

First, planning and evaluating acquisition hardware purely using geometrical coverage does not give the full information to inspection planning experts. The visibility of product surface and defects highly depends on texture, reflectance, light and camera properties which have to be simulated in such a virtual inspection environment. Therefore, this work is extended by integrating a physically-based image synthesis pipeline which simulates light, camera and product surface. As such it enables photorealistic image synthesis enabling estimation of product visibility and coverage with a particular setup of acquisition camera and light. Further, surface texture modeling techniques based on machining knowledge for realistic surface reflection giving the required information on product visibility are presented.

Second, besides acquisition hardware planning, automated inspection systems require robust defect recognition. This can be done either using classical image processing techniques or using machine learning techniques, which becomes prevalent. However, machine learning defect recognition algorithms require a large amount of precisely annotated image data with diversified defects which is often not available due to the rare occurrence of defects or small production batches. Therefore, the synthesis of the defected product surfaces integrated as a part of the virtual surface inspection planning environment represents a solution. In this work, techniques for parametric simulation of geometrical defects on product surfaces as well as pixel-precise, automated annotations are presented. This way, the required amount of realistic product images with diversified defects can be generated for the development of robust defect recognition algorithms.

1.2 Image Synthesis for Virtual Surface Inspection Planning

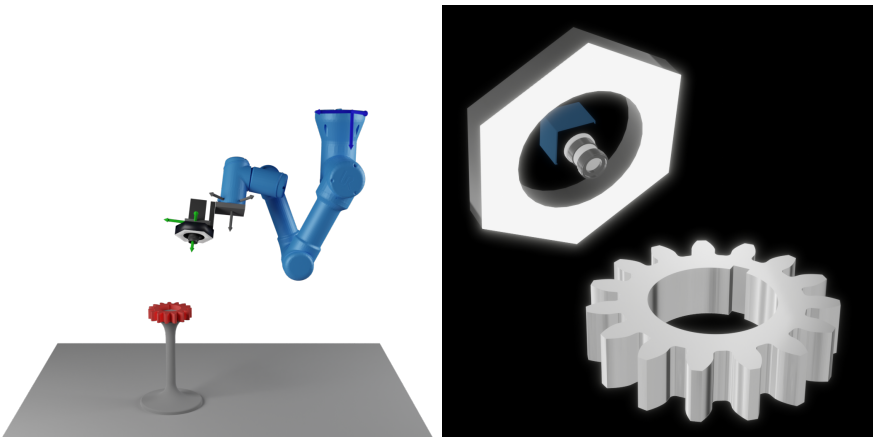


Figure 1.3: Real (left) and virtual (right) inspection environment. Only important elements of a real virtual environment are simulated: camera, light and object.

Virtual surface inspection planning, as discussed in Section 1.1, relies on image synthesis for the configuration and planning of acquisition hardware placement as well as for the development of defect recognition algorithms. Image synthesis using computer graphics requires 3D scene modeling and rendering. The 3D scene represents a virtual environment resembling the real inspection environment as closely as possible (Figure 1.3). Rendering, on the other hand, uses 3D scene information to create synthetic images for a particular view. Realistic image synthesis requires modeling of a 3D scene containing all important elements of a real inspection environment, i.e.,

inspected product, light and camera as well as a physically-based rendering method for image generation (Figure 1.4). The image synthesis background required for this thesis is discussed in Chapter 2.

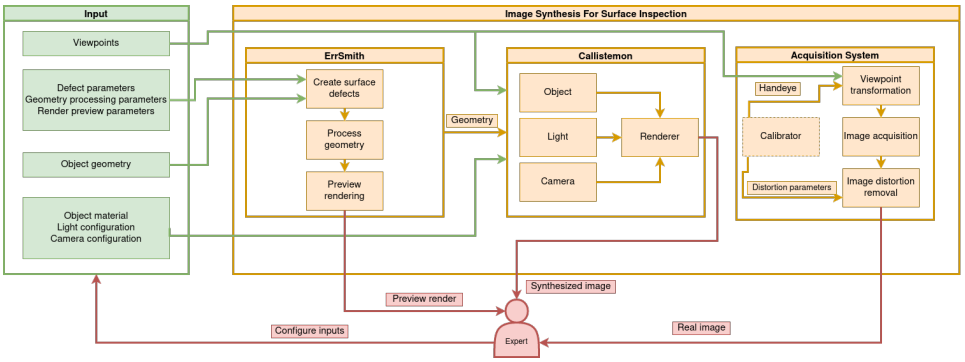


Figure 1.4: Image synthesis for virtual inspection environment as presented in Chapter 4. First, an inspection planning expert provides information about acquisition hardware and inspected objects. Second, defects are simulated and geometrically imprinted. Third, a 3D scene containing simulated light, a camera and an object is rendered to obtain synthetic images. Finally, real images can be acquired for comparison and verification.

Although all elements of such a 3D scene have to be modeled realistically, modeling of product surface texture and surface defects are identified as crucial for virtual inspection planning, making them the focus of this thesis. There are several reasons for this decision. First, the placement of acquisition hardware to achieve the desired product coverage highly depends on the surface response to light. Due to highly reflective materials (e.g., metals) and a wide range of surface finishing textures, the light reflection will significantly vary, which in turn causes varying surface appearance (Figure 1.5). Second, since the goal of the visual surface inspection is to find defects, it is crucial to develop robust defect recognition algorithms and as such are more and more based on machine learning. Defect recognition based on machine learning requires realistic image data of inspected product surfaces containing various surface textures and defects as well as precise defect annotations. Therefore, realistically modeling defect shapes to simulate the correct light scattering on defects is required (Figure 1.5).

In this thesis, both surface texture and defects are modeled procedurally. This approach results in models which can be controlled by a set of parameters, which can further be used to perform automated, realistic and controllable surface texture and defect generation. The procedural modeling background required for this thesis is discussed in Chapter 3.

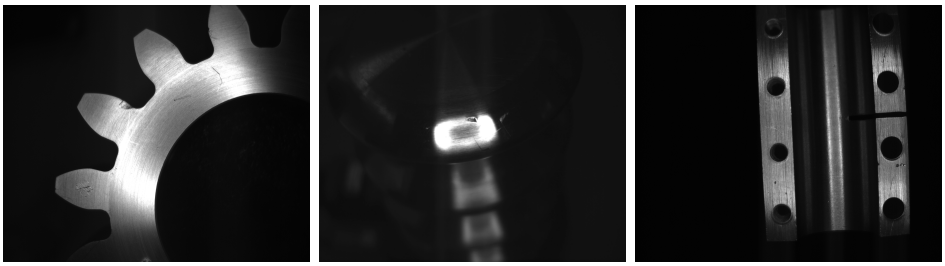


Figure 1.5: Physical samples (gear, spring and clutch) showing different textures and defects

1.2.1 Image Synthesis Overview

Image synthesis using computer graphics is founded on two main steps: *3D scene modeling* and *rendering* (Figure 1.6).

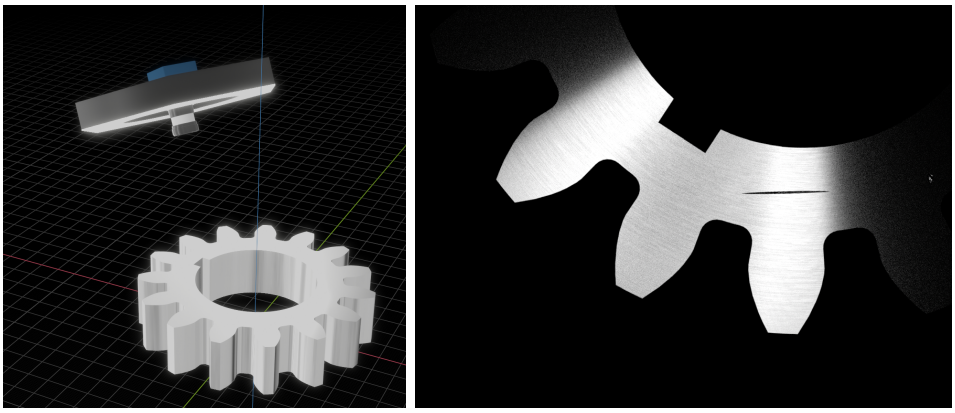


Figure 1.6: Synthesizing image requires modeling a 3D scene (left) and rendering to obtain the final image (right).

3D scene contains elements of a desired virtual environment, e.g., a virtual inspection environment. Specifically, it contains 3D objects, lights and cameras (Figure 1.7) placed in 3D space. A 3D scene is an approximation of the physical environment it aims to represent. Therefore, modeling a 3D scene is a trade-off between computational complexity and the quality of the synthesized image, depending on the application requirements. 3D objects are represented with geometry, material, position, orientation and size in 3D space. Geometry describes the shape of 3D objects. Material is represented with reflectance function and texture which defines how the 3D object will appear when illuminated, that is, how 3D objects interact with light. Thus, the appearance of 3D objects is tightly connected to the light in the 3D scene. Lights, next to emission properties, can also have

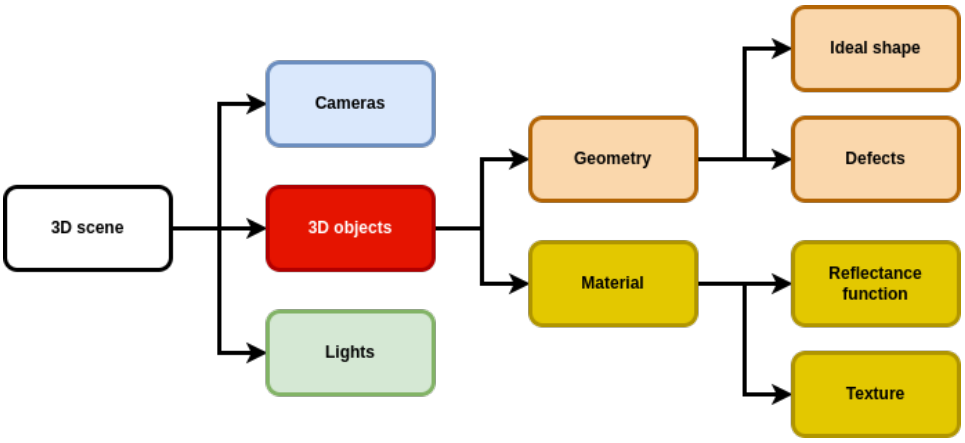


Figure 1.7: 3D scene overview. In this work, the focus is on 3D object surfaces, specifically surface texture and surface defect modeling.

certain geometry, size, position and orientation which highly influence how 3D objects will appear and which parts of objects will be more or less visible. Only reflected light which is captured with a camera will influence the final image. Therefore, parts of a 3D object which will be visible depend on camera properties and its position and orientation in a 3D scene.

That said, it can be seen that all elements of the 3D scene are highly connected and contribute to the final synthesized image. Properties of the modeled 3D objects, lights and cameras can be arbitrary, thus 3D scene modeling depends on a specific application. A Plethora of methods have been developed for modeling each scene element. In Chapter 2 an overview of 3D scene elements relevant to applications in virtual inspection planning is provided.

For the sake of completeness, it is worth mentioning that by including the time component in a 3D scene, it is possible to simulate motion (i.e., animation) and interaction with scene elements. This way, 3D scene elements can move and deform based on user input (e.g., force) or animation rules. In this case, additional material properties are assigned to 3D scene objects describing how they would move and deform, e.g., rigid or soft bodies. Sense of movement is achieved by rendering a series of images (i.e., frames) for each change of 3D scene elements. In this thesis, only rigid, static 3D objects and the single, static positions of the camera and illumination in 3D space around them are considered.

Rendering is a process of creating 2D images from the 3D scene. Rendering can be seen as photographing an object of interest. The resulting 2D image contains 3D objects viewed and illuminated from a particular direction. The rendering process can be decomposed into two steps: visibility

solving and shading. *Visibility solving* uses camera and geometry information of 3D objects to determine which 3D objects are visible from the camera's point of view. Once visible 3D objects are found, their geometry and material as well as light and camera information is used in *shading* step to compute the color. Computed colors are used to construct the final 2D image. 3D scene modeling is highly connected to rendering since its elements must be interpretable by a rendering algorithm. Therefore, it is important to note that 3D scene modeling has to be both usable for human experts as well as interpretable by rendering procedure. Rendering is thus a formalized process and different methods and techniques exist for both solving visibility and shading which will be discussed further in Chapter 2.

Image is the result of the rendering procedure as it will be shortly discussed in Chapter 2. It is a 2D representation of the 3D scene. Each image is taken from a particular camera position with particular 3D scene parameters. The resulting image can be further processed, e.g., color correction or tone mapping.

1.2.2 Procedural Surface Texture Modeling

A particularly important element of 3D scene for virtual surface inspection planning is inspected product surface. Surface properties highly determine its appearance and visibility for particular configurations of light and camera. Surface modeling can be decomposed into modeling bidirectional reflectance distribution function (BRDF) and texture which combined define the surface material. BRDF is a parameterized function describing the amount of surface reflection depending on view and light positions. BRDF describes light behaviour on a microscopic level and state of the art models are based on the microfacet model as discussed by Walter [9] et al. Using BRDF with constant parameters for modeling surface results in an overly smooth and unrealistic response. For this reason, the texture is used to modulate BRDF parameters as well as small-scale geometrical details across the surface. Therefore, the texture is crucial for describing product surface topography, which can result from manufacturing, and achieving a realistic surface appearance.

Texture, when it comes to surface modeling, can be viewed as a function which assigns a certain value to each point of a surface. In practice, texture is either an image or a procedure. The creation of image textures often relies on the usage of painting, photographing, scanning, etc. and thus is extremely useful for depicting complex details in an intuitive and artistic approach. However, when represented as images, textures can not be varied easily and fast. Further, image textures have a fixed resolution. This poses a problem since zooming in on a surface, on which the image texture is applied, reveals a pixelized texture pattern. Also, the surface area that the image texture can cover without obvious repetitions or seams is limited. Finally, mapping

image texture on complex 3D objects often requires manual inspection to reduce repetition and stretching artefacts.

When it comes to visual surface inspection planning, the goal is to have the possibility to apply a wide number of physically realistic textures to a wide number of geometries. Since the textures should provide the possibility of appearance variation and may not contain tiling artefacts or pixelization effects, the use of image textures is largely not considered suitable. Given that texturing is only one step in a larger inspection planning pipeline, the process should require minimal manual work and not assume the user to have expert background knowledge. That said, the focus was put on procedural texturing.

Procedural texturing is an umbrella term for a number of techniques to create texture using an algorithm (Figure 1.8). This way, the resulting texture contains certain advantages over image texturing. First, the procedural texture is parameterized enabling the generation of diversified patterns only by tuning the parameters. Second, since the texture is defined algorithmically, it can be evaluated for any surface point no matter the viewing distance or position. This enables high-resolution textures without repetition artefacts.

Procedural texturing can be decomposed into *authoring* and *generation* steps as discussed by Deguy [10]. In the authoring step, modeling of a procedural texture is performed. On the other hand, in the generation step, the evaluation of the procedural texture model with specific parameters is performed resulting in a particular texture instance. Procedural texturing models can be explicit or implicit. The *explicit model* is evaluated in a raster image which is then mapped on a 3D object. The *implicit model* is evaluated during rendering. Modeling of a procedural texture is often constructed using a combination of regular and irregular patterns. Regular patterns are often periodic mathematical expressions such as sine wave. Irregular patterns are often produced using noise functions such as Perlin noise [11] which generates pseudo-random values with desired properties such as small changes in value for small differences in input. A particular downside of implicit procedural models is the complex authoring phase when it comes to more complex patterns. On the other hand, authoring explicit procedural texture models when it comes to more involved patterns is much more tractable. This is because explicit models allow accessing any point in the domain where the texture lives, e.g., pixels of the raster image, while implicit models must answer random rendering queries for arbitrary points in 3D space, e.g., intersection point during rendering. However, the implicit procedural model offers far more flexibility compared to the explicit model since it is directly queried by a rendering engine eliminating the complexity of of image mapping. Nevertheless, explicit models enable fast and controllable creation of image textures, not depending on manual work such as painting or re-scanning. Procedural texturing is crucial for this thesis,

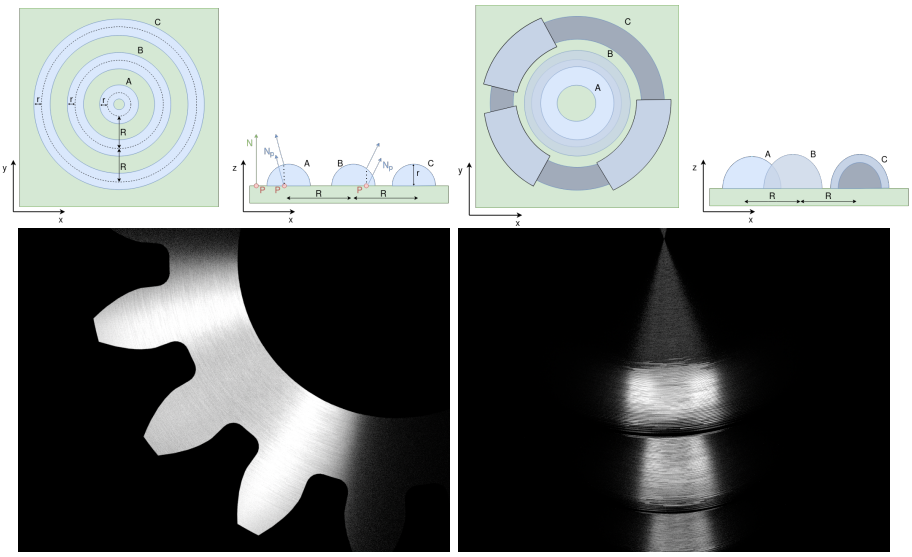


Figure 1.8: Procedural texturing as it will be presented in Chapter 3.1. The diagrams above show the concept of a procedural texturing model for circular brushing which is implemented algorithmically. This procedural texture is used in synthesized objects below for simulating surface topography resulting from machining.

therefore, it is further discussed in Section 3.1.

1.2.3 Procedural Geometrical Defects Modeling

An important task of virtual surface inspection planning is the development of defect recognition algorithms for automated visual surface inspection. Robust defect recognition more and more relies on machine learning. As such it requires a large amount of realistic product images containing surface defects with precise defect annotations. Therefore, alongside surface texture, it is crucial to model surface defects.

Defect modeling is particularly challenging for several reasons such as defect variety, defect unpredictability and defect complexity. Defects are unpredictable in their size, shape and position and can have different causes such as mechanical or chemical. This means they could appear as geometrical imperfections such as dents or scratches but also as material imperfections such as corrosion or liquid stains. In this work, the focus is on geometrical defects such as dents and scratches on metal surfaces. Due to a wide range of factors in product manufacturing and handling, defects often exhibit complex geometrical shapes. The complex defect shapes coupled with varying surface textures affect visibility making defect recognition even harder.

Due to the aforementioned reasons, defects are procedurally modeled as

geometrical alterations, directly on the product surface. Since defects are modeled geometrically, simulation of correct light behaviour, such as inter-reflection and multiple-scattering, is ensured. Further, defect response to light is ensured under varying light and view conditions. Parameterized defecting models enable the controllable creation of diversified defect shapes, sizes and positions over the product surface. Procedural geometry modeling is often used in 3D content creation using general-purpose software. However, general-purpose modeling tools, besides a steep learning curve, require the development involving artistic and computer graphics knowledge, of specialized tools for specific problems such as generating defects. Therefore, the focus is on the development of procedural geometry models which are controlled by physically relevant parameters which require no artistic or computer graphics knowledge.

The development of a procedural geometry model takes place in the authoring phase. The evaluation of the procedural model with specific parameters is performed in the generation phase. The procedural model, in the generation phase, can either produce complete geometry before rendering or only the geometry requested by the rendering procedure (i.e., lazy evaluation). In this work, the procedural defecting model is applied to the 3D geometry of a product before rendering. Authoring procedural geometrical defects requires modeling defect positions and their shapes. Defects can occur anywhere on the product surface but again with a certain bias towards specific parts of the product due to its shape and manufacturing process. Therefore, a wide range of defect positioning possibilities with controllable bias must be accounted for. This can be achieved using sampling of product geometry taking into account its shape features (e.g., sharp edge). Different sizes and shapes of defects can be achieved through procedural geometry shape modeling. Dents and scratches can be considered the most common classes and thus are the focus of this thesis. To model dents and scratches, the focus is on parameterized spherical and cylindrical shapes (Figure 1.9). This way, the high diversity of those two classes can be achieved. These shapes represent defect negatives which are then imprinted into the surface ensuring geometrical defect creation. Since the whole defecting workflow is procedural, it is possible to obtain the exact defect information. This information can be, for example, used for automated synthesis of precise defect annotations as well as for parametric definition of dataset content (e.g., what kind of defects are placed where). Since procedural geometry modeling for defects is crucial for this thesis, it is further discussed in Section 3.2.

1.3 Problem Areas and Contribution

Work presented in this thesis builds on the virtual inspection planning introduced by Gospodnetic [3]. Gospodnetic introduced a virtual environment for

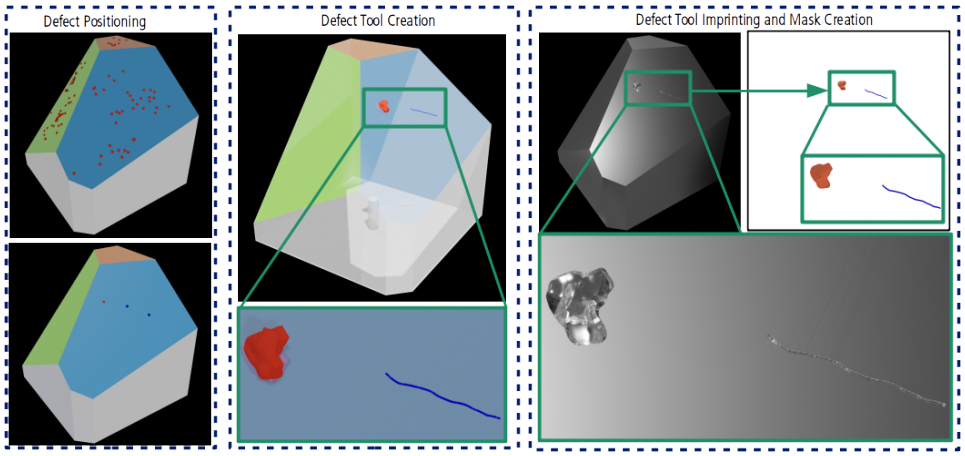


Figure 1.9: Procedural defecting workflow as it will be presented in Chapter 6. First, defect positions are computed. Second, defecting tools are created and placed. Finally, defects are geometrically imprinted and additional geometrical information is generated for rendering annotations.

configuration and placement planning of acquisition hardware for obtaining the required geometrical product coverage. Although assessing geometrical product coverage virtually is helpful for inspection planning experts, the actual product coverage heavily depends on the surface response to light. To obtain realistic product surface visibility, besides its geometry, it is crucial to simulate its surface material and texture as well as its interaction with light. Once the realistic light response of the surface is simulated, the inspection planning expert can fully virtually perform the configuration and placement planning of acquisition hardware for achieving the required coverage. This thesis is further motivated by the need for the creation of robust machine learning defect recognition algorithms which are crucial for automated visual surface inspection. The main obstacle is the lack of physical products with varying textures and defects. A solution lies in image synthesis. Having automated, controllable and realistic image synthesis of products with various surface textures and defects would make the development of robust, machine-learning defect recognition algorithms much more tractable.

Motivated by the aforementioned discussion, the focus of this thesis is to provide a set of image synthesis tools, based on computer graphics: 3D scene modeling and rendering, for solving the problems regarding acquisition hardware planning to obtain the required coverage and development of defect recognition algorithms. **The first contribution** consists of requirements for image synthesis for surface inspection and the development of an image synthesis pipeline for visual surface inspection. The presented pipeline enables inspection planning experts to define product, camera, light and desired defect information which is used for creating the virtual inspec-

tion environment. The virtual environment is represented as a 3D scene containing both defect-free and defected inspection products, illumination and a camera. The pipeline enables realistic image synthesis using physically based rendering methods. This way, inspection planning expert can assess the surface visibility for the particular configuration and placement of acquisition hardware virtually, without computer graphics knowledge. **The second contribution** consists of texture modeling requirements for visual inspection and procedural texture synthesis models for modeling realistic product surfaces satisfying those requirements. Introduced models are parametric, enabling automated, controllable and realistic image synthesis of varying product textures common in machining. Resulting parameterized texture synthesis models enable inspection planning experts to synthesize images containing products with surface textures resembling real surfaces resulting from machining. This way, it is possible to obtain an estimation of surface visibility since the light response from the simulated surface resembles the light response from real machined surfaces. **The third contribution** consists of requirements for defects modeling and procedural models for generating geometrical defects on inspected product geometry satisfying those requirements. Resulting parameterized models enable the controllable generation of defected 3D product instances for image synthesis. This way, together with the second contribution, it is possible to perform verification of defect visibility for specific setup of acquisition hardware. Furthermore, presented models enable automated and controllable generation of an arbitrary amount of defected product instances with pixel-precise annotations, thus enabling the synthesis of training-ready datasets required for the development of machine learning defect recognition algorithms. That said, this work contributes to the following problem areas:

- Image Synthesis for Surface Inspection (Chapter 4)
- Procedural Texture Synthesis for Surface Inspection (Chapter 5)
- Procedural Defect Modeling for Virtual Surface Inspection (Chapter 6)
- Synthetic Data for Defect Segmentation on Complex Metal Surfaces (Chapter 7)
- Image Synthesis for Machine Vision in Metal Surface Inspection (Chapter 8)

1.3.1 Image Synthesis for Surface Inspection

Product image data containing varying surface textures and diversified defects, is required for both inspection planning and development of defect

recognition algorithms. Due to the lack of physical product samples, image synthesis comes as a logical solution.

Photo-realistic image synthesis is one of the main goals of computer graphics. 3D modeling and rendering required for realistic image synthesis are predominantly researched and developed for applications in the film and game industry. Although those methods result in an impressive amount of realism, they are developed to empower the artistic creative process. Therefore, the synthesis of realistic imagery is heavily dependent on the artist's knowledge, experience and iterative manual work.

As a solution to problems regarding the lack of physical samples and artist-centric image synthesis, a set of requirements which has to be satisfied for image synthesis in the visual surface in inspection planning is first presented. Furthermore, an image synthesis pipeline for surface inspection, suitable for inspection planning experts is presented. Based on the information on the inspection planning environment (inspected object, illumination and camera), desired defects and acquisition plan, the pipeline simulates the inspection environment as a 3D scene and performs physically-based rendering for generating realistic images of both correct and defected inspected object. With the presented pipeline, inspection planning experts can completely virtually assess the acquisition hardware configuration and placement for achieving the required coverage. Furthermore, the presented pipeline allows for synthesizing an arbitrary amount of defective image data with defect annotations for developing machine-learning-based defect recognition algorithms. Contributions are discussed in Chapter 4.

1.3.2 Procedural Texture Synthesis for Surface Inspection

Configuration and placement planning of acquisition hardware highly depend on product surface reflection. The crucial element of realistic object surface appearance in synthesized images is surface material. Modeling of surface material is predominantly researched and developed for achieving realism in terms of perceptual appearance. Therefore, existing methods are dependent on artistic experience for describing the realistic distribution of material and its properties across object surface. Numerous advanced methods and techniques are developed for such purpose and available in software such as Substance Painter [12] for highly realistic texture modeling. However, existing methods require a lot of experience, artistic skills and manual work to achieve realistic results. Furthermore, existing texture synthesis models do not take into account the physical and manufacturing parameters of surfaces which determine texture structure.

As a solution to this problem, a set of requirements that have to be satisfied for texture synthesis in visual surface inspection planning is first presented. Further, several texture synthesis models capable of producing surfaces often found in machining and suitable for surface inspection are

presented. Presented models are based on procedural modeling techniques and incorporate surface metrology and manufacturing knowledge to represent machining surfaces. Texturing models are parametrized and integrated into the image synthesis pipeline, enabling inspection planning experts to automatically create realistic textures in synthesized images only by tuning the parameters. With the presented models, inspection planning experts can estimate surface visibility for particular configurations and placement of acquisition hardware. This contribution is key for assessing the desired product coverage completely virtually. Contributions are discussed in Chapter 5.

1.3.3 Procedural Defect Modeling for Virtual Surface Inspection

Automated inspection heavily relies on robust, machine-learning defect recognition algorithms. Defects appear as complex geometrical shapes exhibiting complex light scattering, varying under different view and light conditions. Due to their complexity, the development of defect recognition algorithms requires diversified defects as well as precise annotations. Motivated by the lack of defected physical product samples, the focus is put on defecting an inspected object in a 3D scene.

Generating complex geometrical shapes in virtual worlds is widely researched. Often, a solution to this problem is based on procedural techniques which are capable of creating an immense amount of detail and variety by only tuning the parameters. Unfortunately, existing procedural models require a lot of computer graphics and artistic knowledge as well as significant experience followed by repetitive workflow for achieving the desired results.

To solve this problem, requirements which have to be satisfied by defecting models in visual surface inspection planning are first presented. Furthermore, parametric methods for defect placement, defect tool modeling and defect tool imprinting are presented. The presented methods automatically generate information needed for synthesizing defect annotations next to realistic images. With the presented methods, it is possible to generate an arbitrary amount of images containing defected product samples with pixel-precise annotations. These methods enable assessing defect visibility for particular configurations and placement of acquisition hardware. Finally, the presented methods represent a solution to the lack of products with various defects, required for the development of defect recognition algorithms. Contributions are discussed in Chapter 6.

1.3.4 Synthetic Data for Defect Segmentation on Complex Metal Surfaces

Visual inspection of metal surfaces poses a particular challenge due to high reflectivity making surfaces appear either too bright or too dark for defect recognition. Further, complex surface textures on metal surfaces often cause complex anisotropic reflections which in turn cause low visibility of surface defects. The complexity of light behaviour on metal surfaces is making the development of defect segmentation algorithms particularly challenging. Further, robust defect segmentation algorithms rely on machine learning approaches which require diversified defect samples providing a complete understanding of all the possible defect characteristics and occurrences on the production line. Existing datasets for metal defect segmentation are sparse due to a variety of reasons such as small production batches or low frequency of defects. Further, existing datasets often contain planar metal surfaces which are hard to generalize to more complex, curved surfaces.

To solve this problem, introduced methods for procedural texture and defect modeling are used to generate synthetic datasets which are representative of their real counterparts. The particular focus is on metal parts and thus a representative synthetic dataset, for the domain of multi-view inspection of a complex metal object is introduced. Simulated metal parts consist of curved surfaces, with parameterized textures and geometrical defects enabling the generation of realistic, diversified and precisely annotated images. Contributions are discussed in Chapter 7.

1.3.5 Image Synthesis Pipeline for Machine Vision in Metal Surface Inspection

Advances in manufacturing make defect recognition tasks more and more involved and complex due to at least two main reasons. First, improvements in production processes result in less frequent defects as well as defects which are harder to recognize. Second, increased customization capabilities in production result in a large variety of surfaces (e.g., surface materials and finishings) making it harder to adapt the defect recognition algorithms. Since tasks in industrial environments are very specific and there are almost non-existent publicly available datasets, obtaining the required data is challenging. Therefore, a suitable, well-balanced dataset must be created for each new inspection task which is costly, time-consuming and challenging.

Synthetic image data generation provides control over content and diversity for dataset creation, speeding up the process and reducing its cost. As such, it has been used in several domains for machine vision such as human-oriented, autonomous driving and robotics. This thesis introduces a pipeline for industrial image synthesis enabling controlled data generation of physically correct synthetic images for surface inspection planning domain.

The pipeline extends the methods introduced here for image synthesis and defect modeling for surface inspection. As such it introduced the following contributions: (1) extension of image data synthesis introduced in Chapter 4, (2) procedure for measurement of surface and defects topography, (3) separation of surface modeling into micro, meso and macro geometry scales, (4) requirements for stochastic geometry modeling based on the measurements, (5) parameters for controllable synthetic dataset generation and variation, (6) methodology for evaluating the quality of synthetic data and (7) publicly available dual dataset for defect recognition. For the purpose of this work, the focus is on milled and sandblasted metal surfaces without coating, containing dent and scratch defects. While the measurement and modeling process can be performed for both texture and surface defects, in this work, the focus is on the texture modeling approach. Contributions are discussed in Chapter 8.

Chapter 2

Image Synthesis Background

Image synthesis using computer graphics relies on 3D scene modeling and rendering. Therefore, in this chapter, the background knowledge, relevant to this thesis, regarding 3D scenes, rendering and images is provided.

The 3D scene consists of 3D objects, lights and cameras which reside in certain a 3D space. Once modeled, a 3D scene is used for rendering which creates viewable 2D images. Finally, synthesized images can be processed before viewed on a display device or used for further computations.

2.1 3D space, Transforms and Scene Graph

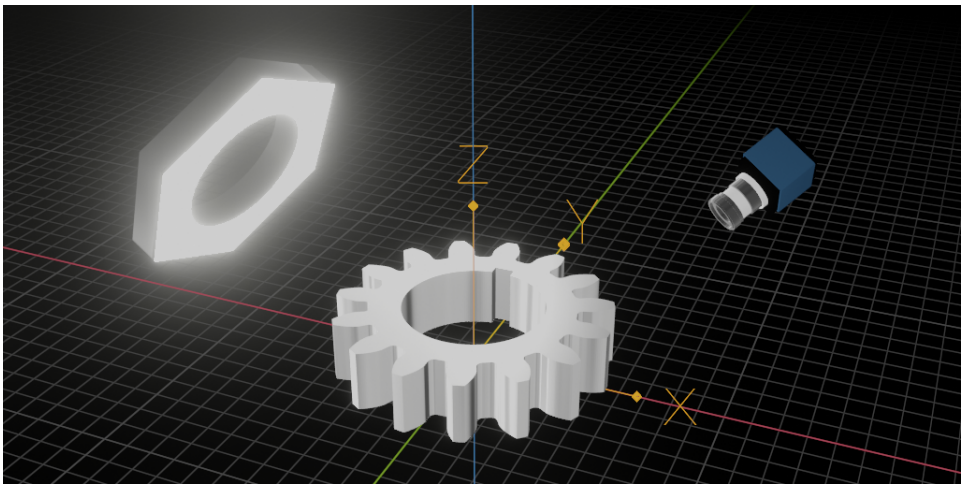


Figure 2.1: 3D space containing light, camera and 3D object placed in world coordinate system.

All 3D scene elements; 3D objects, lights and cameras are defined in a certain 3D space (Figure 2.1). To explain, 3D object geometry, position, orientation and scale are defined with a set of 3D points and 3D vectors.

Similarly, light geometry, its location, orientation and scale are defined with a set of 3D points and 3D vectors. Finally, camera location and orientation are defined with a set of 3D points and 3D vectors. That said, points and vectors must be defined relative to a certain 3D coordinate system (Figure 2.1).

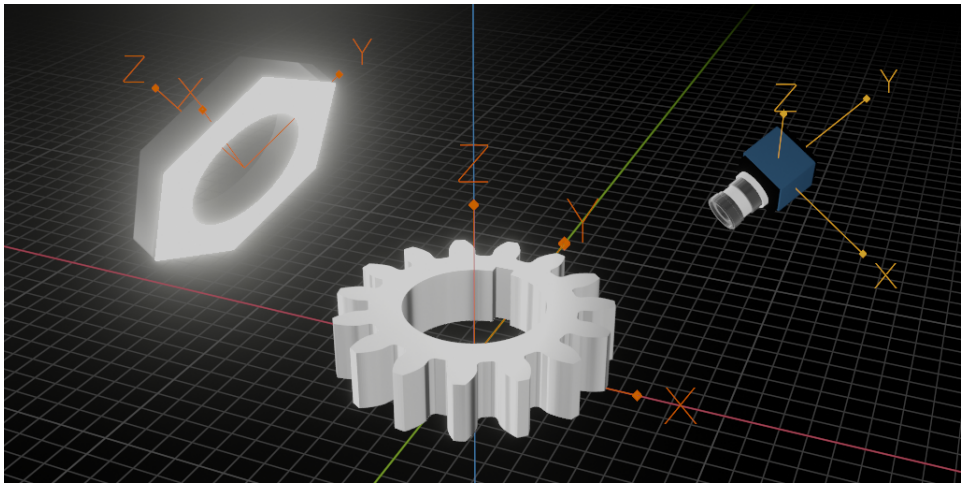


Figure 2.2: All 3D scene elements have a local coordinate system relative to the world coordinate system.

In computer graphics, the most often used 3D coordinate system is Euclidean. The *Euclidean 3D coordinate system* is defined with *origin* and *basis*. *Origin* is defined with a 3D point, while *basis* is composed of three orthogonal vectors that define the X, Y and Z axes. Although the Euclidean coordinate system is most often used, other coordinate systems, e.g., polar, can be used in certain scenarios (e.g., *shading*) for more tractable computations. Since the Euclidean 3D coordinate system is defined with a set of points and vectors, there is a need for a standard coordinate system. For this purpose, a *world coordinate system* is defined. All other, i.e., *local coordinate systems* are defined relative to *world coordinate system* (Figure 2.2). Therefore, local coordinate systems of 3D scene elements, such as object or camera coordinate systems, are defined relative to the world coordinate system.

To translate, rotate or scale 3D objects, lights and cameras across a 3D scene, *transformations* are used. Transformations are represented as 4x4 matrices which take 3D points and vectors in *homogeneous* form, defined with respect to one coordinate system and map their coordinate values to another coordinate system. Basic transformations are *translation*, *rotation* and *scaling*. Often, more advanced and specialized transformations such as *look-at transform*, *Euler transform*, *quaternions* or *projections* (e.g., perspective projection) are used since they are more practical or more robust

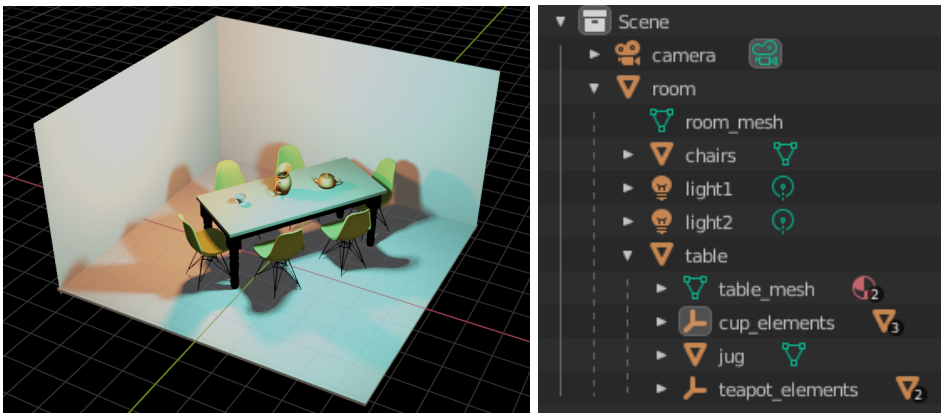


Figure 2.3: Example of a scene graph. The scene represents the root node where the camera and room are its children. Further, the room has a table for child which has cup, jug, teapot for children.

for specific applications as discussed by Lengyel [13].

More complex 3D scenes contain a large number of 3D objects and lights as well as their respective transforms. To simplify 3D scene creation, storage and rendering, *scene graphs* are used. From the user side, scene graphs enable easier authoring of complex scenes. On the other hand, it represents a clear structure which is traversed during rendering. Since the spatial arrangement of 3D scene elements has an inherent tree-like structure, scene graphs are implemented as a tree data structure (Figure 2.3). The root node is the starting point for the whole 3D scene. Internal nodes organize a scene into a hierarchy. A wide range of internal nodes exist and the most often ones are transformation nodes. Leaf nodes are actual scene elements such as 3D objects, lights and cameras.

2.2 3D Objects: Shape Representations

3D object shape is represented using geometry. From the modeling side (i.e., user authoring side), geometry is used to obtain the desired 3D object shape. From the rendering side, geometry is used for solving the *visibility problem*: which objects are visible from the camera's point of view as well as which objects are visible to each other (i.e., no surfaces obstructing the view). Further, geometry plays an important role during the *shading* step during rendering since it gives information on where a 3D object is placed relative to the camera and light. The crucial geometrical information required for shading is *surface normal* which describes the surface orientation. The role of geometry in rendering will be further discussed in Section 2.6.

Due to various applications of computer graphics aiming to represent different shapes, a wide range of object shape representations were devel-

oped and are used in computer graphics. Some shape representations are more suitable for modeling, while others are more tractable for rendering. Therefore, the choice of shape representation depends on the application, 3D modeling workflow and rendering approach. Thus, efficient methods for mapping *modeling-friendly* shape representations to *rendering-friendly* shape representations were also developed.

Since the focus of this thesis is on metal rigid products, the interior of the product is not relevant. In this case, *surface representations* such as *polygon mesh*, that is, *triangulated mesh* are favourable, as discussed in the next section. Further, products are often modeled using CAD programs relaying on *parametric surface representations*, also discussed in the following sections, which can be converted to mesh representations. For the sake of completeness, other, commonly used shape representations are also discussed in the following sections.

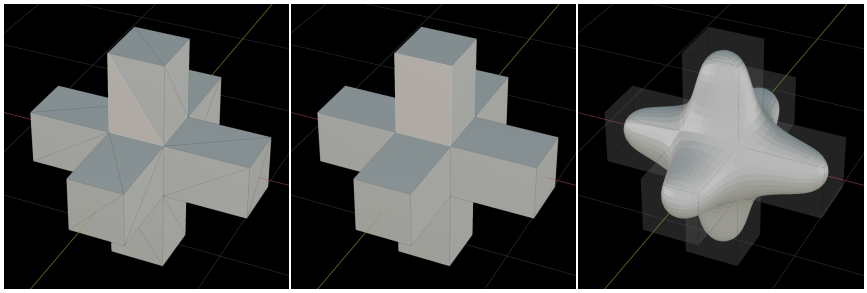
Polygon mesh

Polygon mesh is suitable for representing the surface of a 3D object in 3D space as discussed by [14], ch. 3. Most often used polygonal meshes are triangle meshes (Figure 2.4a). Triangles consist of three vertices making them the simplest type of polygon. Further, triangles turned out as most efficient rendering primitive due to their simplicity and good computational properties (e.g., co-planar property). Triangle mesh can represent a wide range of complex shapes. Therefore, other shape representations are often *triangulated* and rendered as triangle mesh. Triangulation is the process of converting shape representation into triangulated mesh. It is a special case of *tessellation* where shape representation is transformed into a polygon mesh.

Although efficient for rendering, triangulated meshes are not intuitive for modeling complex shapes. For this reason, quadrilateral (quad) mesh polygons are often used and are very convenient for modeling (Figure 2.4b). A Quad is a polygon made of four vertices. An additional advantage of quadrilateral mesh is that it can be efficiently triangulated and rendered as triangle mesh.

Polygons with a number of vertices that go beyond four are not often used since the modeling and rendering complexity increases. Further, such polygons can be constructed using multiple triangles or quads.

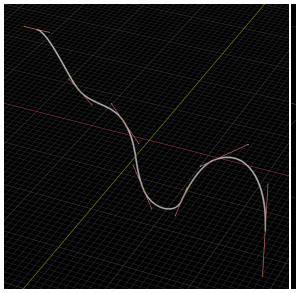
The problem with polygonal meshes is that a smooth surface is only approximated using flat polygons, making surfaces look faceted. Various solutions were developed to reduce this effect. One solution is introduced by Phong [15], where surface normals are interpolated across polygon and used for shading. This method is called *Phong shading* and it only makes the final surface to *appear* smooth without altering actual surface geometry. A different approach to achieving smooth surfaces is based on *subdi-*



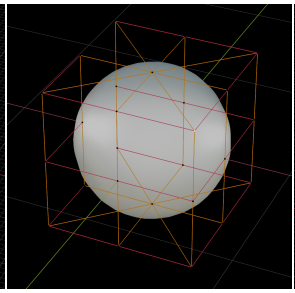
(a) Triangle mesh.

(b) Quad mesh.

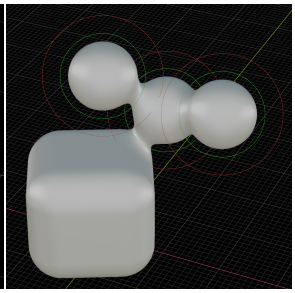
(c) Catmull-Clark subdivision surface.



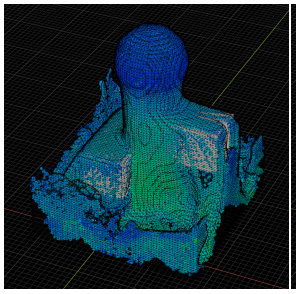
(d) Bezier curve.



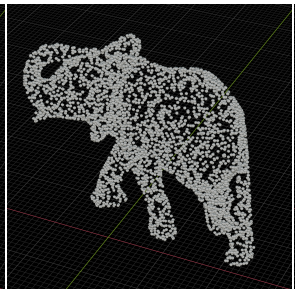
(e) NURBS surface.



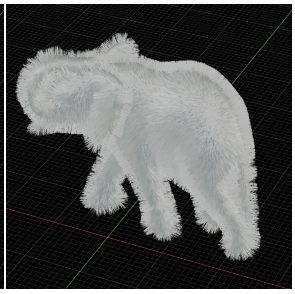
(f) Implicit surface.



(g) Voxel and particles for fluid simulation and visualizuation.



(h) Point cloud.



(i) Particles.

Figure 2.4: Shape representations.

vision surface techniques as discussed by Akenine et al. [1], ch 17. These methods solve the problem by introducing additional polygons for smoother surface approximation. Subdivision surface methods consist of a two-step process: refinement phase and smoothing phase. In the *refinement phase*, new vertices are created and reconnected to create new, smaller polygons. In the *smoothing phase*, new positions for some or all vertices in a mesh are computed. Different strategies exist for both the refinement and smoothing phases yielding different subdivision surface methods. The most widely used subdivision method is Catmull-Clark [16] (Figure 2.4c).

Besides being efficient for the visibility-solving step of rendering, polygon mesh can contain information for the shading step during rendering. Shading information is stored per vertices of the mesh and thus called *vertex attributes*. Vertex attributes can be colors, normals, texture coordinates, etc. Texture coordinates are needed for applying textures which are crucial for realistic shading. Texture coordinates for polygon mesh can be essentially computed in two main ways: *mesh unwrapping* and *texture projection* as discussed by Akenine et al. [1], ch. 6. *Mesh unwrapping* aims to unwrap mesh on a 2D surface while minimising distortion artefacts. 2D coordinates of unwrapped mesh vertices are called texture coordinates. *Texture projection* aims to use simpler shapes such as spheres or cubes which are easily assigned with texture coordinates to project those coordinates on a mesh.

Parametric curves and surfaces

Parametric surfaces are another common type for representing surfaces in 3D space using parametric form, that is, analytical functions. Due to their parametric description, parametric surfaces are compact, controllable, scalable and capable of representing smoother and more continuous geometry shapes. Their advantage compared to polygonal mesh is that they can be represented with few control points. This enables easier modeling and less storage space. The parametric form is evaluated on the fly generating scalable geometry. Parametric surfaces are often used for CAD modeling in engineering and architecture as well as modeling curved, smooth and organic surfaces.

Parametric surfaces are two-dimensional extensions of parametric curves which are one-dimensional analogues. Parametric curves are often used for representing thin objects, e.g., hair or for animation purposes for controlling the movement. The simplest and most intuitive are Bezier curves as discussed by Akenine et al. [1], ch. 17, which are based on the idea of repeated linear interpolation described by de Casteljau algorithm and compactly described with Bernstein form (Figure 2.4d). Repeated linear interpolation is done between two endpoints and controlled with n control points. For one control point, the result is a linear Bezier curve, that is, linear interpolation. For two control points the result is a quadratic Bezier curve. For three

control points, the result is a cubic Bezier curve. Therefore, the degree of the Bezier curve is n if $n + 1$ points are used. Bernstein form enables the decoupling Bezier curve as the sum of multiplications of basis function and control points. The basis function describes the influence of each control point as parameter changes. Further, it defines the properties of the curve such as that the whole Bezier curve will be located in a convex hull of control points. For efficient computation, the matrix form of the Bezier curve can be used. The most optimal is the cubic Bezier curve which can describe complex forms such as *inflection* and which has low computational complexity. In practice, multiple Bezier cubic curves are concatenated to form a larger spline called a piecewise Bezier curve, connecting multiple input points. Joining curves must be done with care so that the desired *continuity* of the resulting curve is achieved as discussed by Akenine et al. [1], ch. 17. Therefore, *positional continuity,* C^0 requires curves to be connected at the same point often called *joint*. Next, *velocity continuity,* C^1 requires that derivations of connected curves in any point must be continuous. Finally, *acceleration continuity,* C^2 requires that first and second derivations of connected curves in any point must be continuous. Similarly, curve continuity can also be described using *geometrical continuity*.

In practice, more controllable *cubic Hermite* curves discussed by Akenine et al. [1], ch. 17 are used. The cubic Hermit curve is described with starting and ending points and two controllable tangent vectors at each point. Cubic Hermite curve can also be described using Bernstein form for analysis of Hermite basis function and matrix form for efficient computation. Multiple cubic Hermit curves can be combined into the assembly of curves connecting input points. The main design decision for combining curves falls on the computation of tangents at each point. An efficient method for computing tangents for control points is Kochanek-Bartels which offers *tension* and *bias* parameters for shaping the curve. The special case of the Kochanek-Bartels method results in a widely used Catmull-Rom spline.

In practice, the most widely used is cubic B-spline (Akenine et al. [1], ch. 17) which can be intuitively understood from the description given for the Bezier curve. Two main types are uniform and non-uniform which define the spacing of control points. Rational B-spline, a generalization of B-spline, is used to overcome the limitation of describing certain shapes such as circles. Finally, in practice, the most general and flexible curve description is non-uniform, rational B-spline (NURBS).

The discussed properties of parametric curves can be extended to *parametric surfaces*. Common parametric surface types are Bezier surfaces and B-spline surfaces. *Bezier surface* can be explained using repeated bilinear interpolation forming a patch described with de Casteljau's algorithm. Similarly, as for curves, the Bezier surface can be analytically defined using the Bernstein form. In practice, the bicubic Bezier surface is used due to computational efficiency and representational capabilities. A Bicubic Bezier

surface is a building block which is concatenated to create more complex surfaces. Concatenating Bicubic Bezier surface must be also taken with care to fulfil the continuity requirements discussed for concatenating curves. Bezier surface inspired further methods for representing smooth surfaces such as *Bezier triangles*, *Point-Normal (PN) triangles* and *Phong tessellation* as discussed by Akenine et al. [1], ch. 17. In practice, most general and flexible parametric surface representations are *B-spline surfaces* discussed by Akenine et al. [1], ch. 17. Bicubic B-spline surfaces are used to form composite surfaces. Further generalizations resulted in state-of-the-art *Non-uniform rational B-spline (NURBS) surface* (Figure 2.4e).

When it comes to rendering parametric surfaces, two main approaches can be taken. The first approach is to use ray-tracing-based rendering and to analytically define ray-surface intersection. The second approach is to tessellate the parametric surface and render it as a triangle mesh. In this case, the trade-off between the quality of tessellation and the speed of rendering must be decided on.

Implicit surfaces

Implicit surfaces represent object surfaces in 3D space, using a signed distance function (SDF) discussed by Soderlund et al. [17]. SDF is equal to zero if the point is on the surface, less than zero if the point is inside the surface and greater than zero if the point is outside of the surface. Simple SDF shapes, such as spheres, cubes and cones can be described easily with a single expression. Describing complex shapes with a single SDF is rather complex. To construct arbitrary complex shapes, simple SDF shapes are combined using operations such as addition and subtraction into more complex shapes. This way of modeling is called *constructive solid geometry* (CSG) and it is often used in CAD programs [18]. Various approaches for merging SDF shapes exist. One often used approach is "melting" shapes into each other giving a very well-known representation called *metaballs* (Figure 2.4f). Metaballs are extremely useful for modeling blobby, organic shapes.

When it comes to rendering implicit surfaces, there are two main approaches. One is based on ray-marching: tracing rays from a camera, measuring the distance to the closest implicit shape via sphere shape and moving on the ray based on obtained distance [19]. Another approach is to perform tessellation via marching cubes [20]. First, the 3D surface described with implicit representation is partitioned into cells. Each corner of each cell is evaluated if lying inside or outside of the implicit shape. Based on the configuration of cell corners lying inside the implicit shape, the polygon is constructed. The resulting polygonal mesh can then be efficiently rendered.

Voxels

Voxels, as opposed to polygonal meshes, parametric and implicit surfaces, are used to describe both the surface and interior, the *volume* of the 3D object as discussed by Museth et al. [21]. One *voxel* is simply a cube in 3D space and it represents a volume of space. To represent complex objects, a uniform grid of voxels is used, where each voxel can, at simplest, contain binary information if it is inside or outside of a 3D object. Such information is suitable for opaque objects. Further, information stored per voxel can be extended with, e.g., density or opacity information which enables visualization of an object's interior. Therefore, voxels are used in applications concerned with the volume of an object. Further, this representation is particularly useful for fluid simulations such as water, smoke or clouds (Figure 2.4g).

When it comes to rendering voxel representation, two main rendering approaches are used. The first approach is to perform tessellation and render the resulting polygonal mesh. The second approach is to trace rays from the camera, that is, perform ray-marching. It is worth mentioning that large voxel grids are memory expensive and that often not all voxels in a voxel grid are relevant for object representation. Therefore, memory-efficient sparse-voxel representations, e.g., sparse voxel octrees are used as discussed by Laine et al. [22] and Museth et al. [21].

Point cloud and particles

Points cloud representation describes 3D object shapes using a large number of points in 3D space. Point cloud representation is often used for visualizing scanned real-world objects as discussed by Griwodz et al. [23]. This way, complex 3D models can be visualized without the need for costly surface reconstruction or triangulation. Point clouds represent a flexible and efficient alternative to parametric or polygonal mesh surface representations.

Point-based representation of shapes is further efficient for unified physical simulations. As discussed by Macklin et al. [24], point-based representation enables unified rigid, soft and fluid simulation.

Points are fundamental to *particle systems* introduced by Reeves [25]. Simulating different behaviours of points, i.e., particles, through time and assigning them geometrical shapes and textures enables describing various phenomena such as dust, water, smoke, etc.

When it comes to rendering, points are often represented as small 3D shapes (e.g., spheres or capsules) or planar elements (e.g., circles or quads) which can be efficiently rendered as discussed by Pfister et al. [26] and Rusinkiewicz et al. [27].

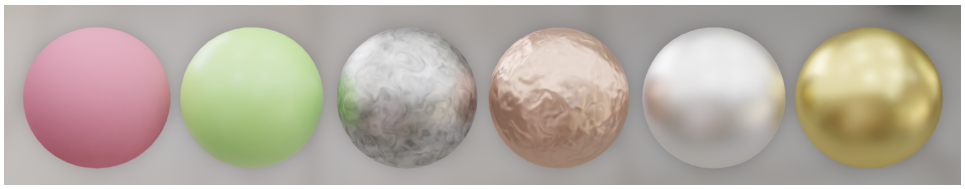


Figure 2.5: Objects with same geometry but different materials.

2.3 3D Objects: Material Representations

Geometry in 3D space enables defining the shape of 3D objects and provides information for determining visibility during rendering. On the other hand, material describes how light interacts with 3D objects and thus defines their appearance. The material description is used for shading computation during rendering. Material information, for example, describes how an object's surface reflects incoming light. Light reflected from objects into the camera is responsible for forming the image. Therefore, material defines how objects will appear in the rendered images. Based on previous discussion, it can be concluded that material modelling is crucial for photo-realistic image synthesis.

In computer graphics, material modeling is decoupled from object geometry modeling, which enables the creation of *material libraries* which can be applied on arbitrary 3D objects (Figure 2.5). Material modeling represents an important, thus rich and complex topic in computer graphics developed over several decades. In this section, a short overview is provided, focusing on aspects important to this thesis.

2.3.1 Material Appearance Observation

Material modeling starts with the observation of its appearance. The goal is to determine which characteristics are responsible for the object's appearance, what is the role of material for appearance and to determine which features make each material look different than other materials. Looking at different objects (Figure 2.6) it can be noticed that object appearance generally depends on the object's shape and material as well as illumination and sensor (e.g., camera position and properties). Isolating material characteristics is required for modeling but is not always easily done or completely possible. For example, without light, it is not possible to see objects or their material and thus properties of light influence how materials appear. The following material characterization is based on human perception as discussed by Dorsey et al. [28]:

- **Materials are characterized by color**. Spectral variations of light reflected from objects into the eye are perceived as color and brightness of the object.

Figure 2.6: Various objects exhibiting a wide range of appearances.

- **Materials are characterized by directional effects**. Variations of light scattered from object's surface account for directional effects. For describing directional effects, terms such as specular, diffuse, glossy, matte, transparent or translucent are used. An example specific to machining is surface finishing which can be highly polished and rough surfaces.

- **Materials are characterized by texture.** Any kind of spatial visual variation on an object's surface on a much smaller scale than the size of the object (that is, its geometry) but a much larger scale than the wavelength of light accounts for texture. Perceived variations are due to spatial variation in color or directional effects as well as small-scale geometrical structures such as bumps, dents, pores, scratches, etc. Surface texture is highly varying both spatially and on different scales.

Material models in computer graphics aim to simulate the observed, real-world, material appearance, namely its color, directional effects and texture. In computer graphics, *material modeling* is decoupled into modeling *reflectance function* and modeling *texture* as discussed by Christensen et al. [29].

The reflectance function describes the color and directional effects of a material. *Texture*, on the other hand, describes any kind of spatial variation. As discussed, this spatial variation can be color variation, directional variation or small-scale geometrical details. Further, texture can be viewed

as a function which for each point of the object surface defines parameters of the reflectance function and small-scale geometrical features (e.g., normals) on which the reflectance function is evaluated.

2.3.2 Optics and Reflectance Functions

Modeling the reflectance function is based on optics, a branch of physics dealing with light and its interaction with the material. Here, the discussion is based on notes by Hoffman [30].

Optics for Computer Graphics

Directional and color effects are tightly connected to the light itself. Light is an *electromagnetic transverse wave* as described in *wave optics*. Light characterization as a wave is often too complex for computer graphics modeling and rendering purposes. Therefore, computer graphics often rely on *geometrical optics* where light is represented as rays.

In geometrical optics, light-matter interaction is described with an index of refraction (IOR), a complex number where the real part determines the speed of light in a matter and the imaginary part describes the absorption of light in a matter. Therefore, light travelling through a medium can be *absorbed*, which causes light attenuation and potential change of color (Figure 2.7). Further, rapid variation in IOR causes light *scattering* - splitting in multiple directions but the overall amount of light does not change (Figure 2.7). Finally, matter can *emit* light as is the case with light sources. In the real world, most media both scatter and absorb light to a certain amount defining the appearance of the medium.

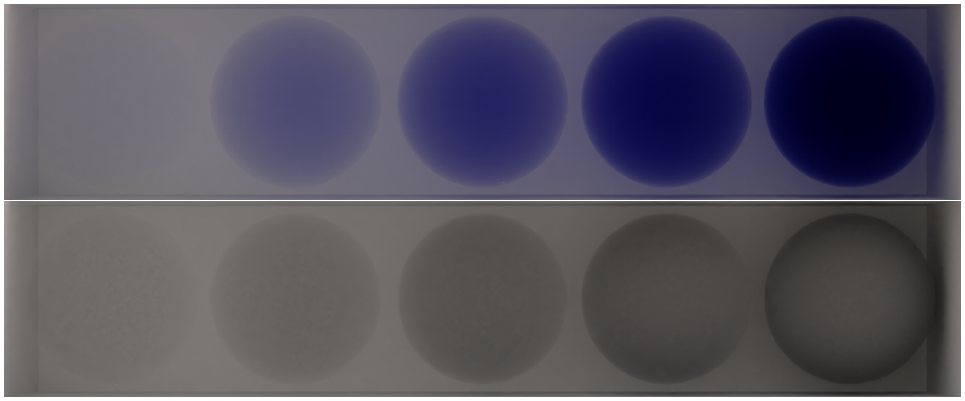


Figure 2.7: Absorption (above) and scattering (below) with varying media density amounts (0.2, 0.5, 1.0, 2.0, 4.0)

.

In the real world, light is travelling not only through one but multiple media with different IOR values. To compute light behaviour between two

media *Maxwell's equations* can be used which, unfortunately, in most cases do not provide analytical solutions. A particular case in which a surface is considered *optically flat* enables deriving a solution which is fundamental for reflectance models in computer graphics.

Optically flat surface is a planar boundary between two media and it is considered flat with respect to the scale of light wavelength. In computer graphics such planar boundary between two media can be represented with a surface, e.g., triangulated mesh. Light, on optically flat surfaces, scatters in exactly two directions: *reflection direction* and *refraction direction* (Figure 2.8). *Light reflection* angle is equal to the incoming angle as stated by the *law of reflection* $\theta_i = \theta_o$ ([14], ch. 8). *Light refraction* angle depends on IOR as stated by *Snell's law*, Eq. 2.1, [14], ch. 8. The amount of light which is reflected and refracted is computed using Fresnel's equations Eq. 2.2, [14], ch. 8. Given the incident θ_i and outgoing light angle θ_o with regards to surface normal and the IOR of both media η_i, η_t, the Fresnel equations specify the material's reflectance for two different polarization states of the incident illumination $r_{||}, r_{\perp}$. Finally, the amount of reflected light F_r can be computed as well as the amount of refracted light F_t using the law of energy conservation.

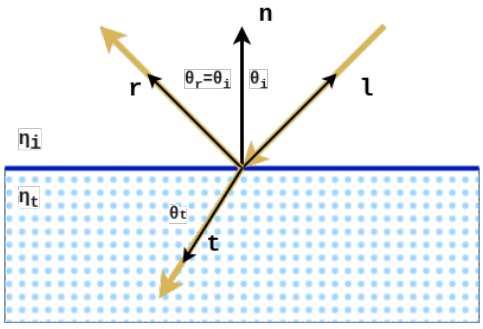


Figure 2.8: Light falling on an optically flat surface scatters exactly in two directions: reflection and refraction.

$$\eta_i sin(\theta_i) = \eta_t sin(\theta_t) \tag{2.1}$$

$$\begin{aligned} r_{||} &= \frac{\eta_t cos(\theta_i) - \eta_i cos(\theta_t)}{\eta_t cos(\theta_i) + \eta_i cos(\theta_t)}, \\ r_{\perp} &= \frac{\eta_i cos(\theta_i) - \eta_t cos(\theta_t)}{\eta_i cos(\theta_i) + \eta_t cos(\theta_t)}, \\ F_r &= \frac{1}{2}(r_{||}^2 + r_{\perp}^2), F_t = 1 - F_r \end{aligned} \tag{2.2}$$

In the real world, surfaces have certain roughness and irregularities and

are rarely optically flat. Therefore, light is not reflected in one direction and thus reflection is not perfectly sharp. An approach to modeling such surfaces is to assume that irregularities are at a scale larger than the light wavelength (thus affecting light reflection) but too small to be observed by the eye (i.e., under pixel). A model of such a surface can be derived by representing a surface as a large collection of tiny optically flat facets for which light interaction can be described using reflection and refraction as discussed. Overall surface appearance is the aggregate result of many facets with different orientations, where each facet reflects incoming light in a slightly different direction. More irregular facets imply a more blurred surface and vice versa (Figure 2.9). This kind of surface model is known as *microfacet model* [31] and represents the basis for state-of-the-art surface reflection models that will be discussed later.

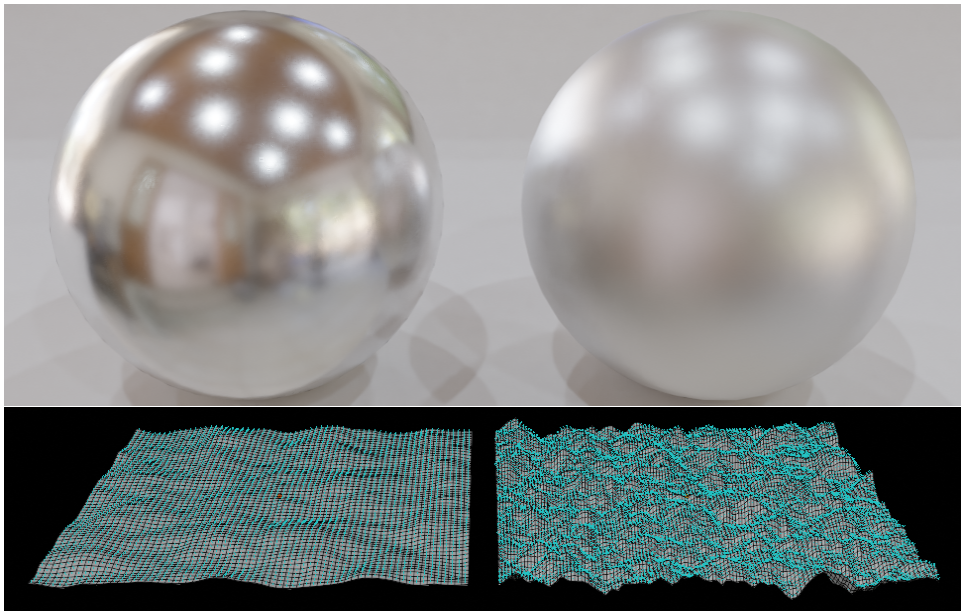


Figure 2.9: Real surfaces often contain irregularities at the microscopic level (not visible by the eye) causing blurred reflection (top). The sharpness of reflection depends on surface regularity (bottom). Irregular surfaces have highly varying normals causing light to scatter in various directions.

Behaviour and amount of reflected and refracted light depends on the material properties. In computer graphics, from the practical point of view, two main classes of materials are often considered: *metals* and *dielectrics* (Figure 2.10). In the case of *metals*, refracted light is immediately absorbed and only reflected light leaves the surface. On the other hand, *dielectrics* exhibit both reflection of light as well as a wide range of absorption and scattering of refracted light. In the case of *opaque dielectrics*, refracted light is sub-surface scattered (SSS), partially absorbed and re-emitted out

of the same surface. Next, in the case of *transparent dielectrics*, refracted light passes through the medium and exits on the surface on the other side. Further, in the case of *translucent dielectrics*, refracted light is scattered, partially absorbed, partially re-emitted and partially transmitted.

In order to reproduce the rich light scattering phenomena various scattering models were developed in computer graphics, which can be separated into *surface scattering* [14], ch. 8 and *volume scattering* [32], [14], ch. 11. For this thesis, surface scattering, specifically surface reflection is particularly important since the focus lies on light interaction with opaque, metal surfaces. However, the discussed principles of scattering and absorption can be applied to volumetric objects like fog or smoke.

Figure 2.10: Various scattering phenomena that can be simulated in computer graphics, from left to right: metal, opaque dielectric, transparent dielectric, translucent dielectric and volumetric.

When it comes to light-surface interaction modeling in computer graphics, the two fundamental light scattering processes that can occur are specular reflection and diffuse reflection (Figure 2.11). *Specular reflection* represents light reflected from a surface. On the other hand, *diffuse reflection* represents light which is re-emitted after being refracted into the surface and undergone partial absorption and SSS. Although diffuse reflection is a result of light interaction below the surface, it is often approximated using only the surface information.

Specular and diffuse reflections are capable of describing a wide range of surfaces and thus are crucial for this thesis. That said, metal surfaces are completely described with specular reflection since diffuse reflection can not occur due to the high absorption of refracted light. On the other hand, dielectrics exhibit both specular reflection (e.g., highlights) as well as diffuse reflection which is responsible for actual surface color. In the following discussion, the focus mostly lies on surface reflection, however, for the sake of completeness, it is worth noting that the principles of surface reflection can be extended for surface refraction and transmission required for modeling transparent objects.

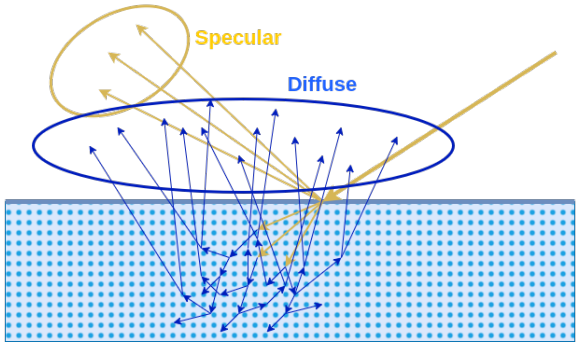


Figure 2.11: Specular and diffuse components of scattered light. Image is recreated based on [30].

Reflectance Functions

In computer graphics, an umbrella term for representing general surface scattering is called *bidirectional scattering distribution function* (BSDF). BSDF can be decoupled into surface reflection - bidirectional reflection distribution function (BRDF) and surface transmission - bidirectional transmission distribution function (BTDF). BRDF can be used for describing metal surfaces or opaque dielectric surfaces. On the other hand, BTDF can be used for describing transmissive dielectric surfaces. For the purpose of this thesis, the focus will be on BRDF models which describe the directional and color effects of metal surfaces.

BRDF describes the surface's response to light. It is a mathematical model approximating light interaction and the microscopic structure of surface material. It considers only light incoming and outgoing above the surface which is determined by surface normal. BRDF is a parameterized function of incoming and outgoing directions (thus named bidirectional). For example, incoming direction can be light direction incoming on the surface while outgoing direction can be view direction - from where the surface is viewed. Therefore, these two directions are often called *view* and *light* direction. Each direction can be described with two polar coordinates giving it a dimensionality of four (Figure 2.12).

BRDF models can be separated into isotropic and anisotropic (Figure 2.13). In the case of an isotropic BRDF, the rotation of incoming and outgoing directions around the surface normal does not affect the BRDF value. Thus, isotropic BRDF can be parameterized with three angles only: θ_i, θ_o and ϕ (Figure 2.12). On the other hand, in the case of anisotropic BRDF, the rotation of incoming and outgoing directions around the surface normal affects the BRDF value. Thus, it must be parameterized with four angles: θ_i, θ_o, ϕ_i, ϕ_o (Figure 2.12). Anisotropic BRDF describes anisotropic surface reflection which is due to particular, often directional, surface mi-

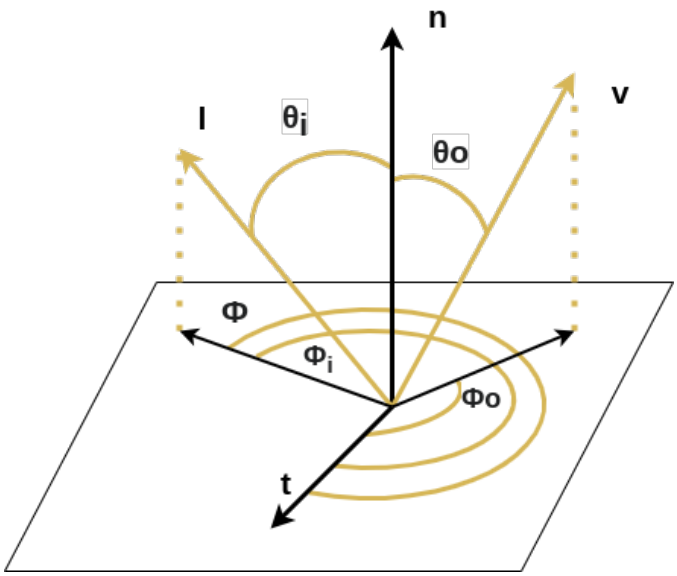


Figure 2.12: BRDF is parameterized with two vectors: view (outgoing) v and light (incoming) l or 4 polar angles: θ_i, θ_o, ϕ_i, ϕ_o. Here, n is surface normal and t is the surface tangent used for anisotropic BRDF. Image is recreated based on [30].

crostructure. The concept of anisotropy is crucial for this thesis since metal machined surfaces are highly reflective and exhibit anisotropic reflection due to surface texture resulting from machining.

BRDF can be interpreted in two ways (Figure 2.14). First, given the incoming ray of light, BRDF gives the relative distribution of reflected light over all outgoing directions above the surface. This interpretation will be particularly useful during modeling and describing different types of BRDF. Second, given the outgoing view ray, BRDF gives a relative contribution of light from each incoming direction. This interpretation will be particularly useful when it comes to rendering since BRDF is a crucial element of the rendering equation which will be discussed later.

For BRDF to be physically plausible, it must respect Helmholz reciprocity and energy conservation as discussed by Guarnera et al. [33]. *Helmholz reciprocity* states that incoming and outgoing directions can be swapped leaving the result unchanged. *Energy conservation* states that reflected light energy cannot be greater than incoming light energy.

BRDF modeling approaches can be roughly separated into:

- **Empirical approaches** are modeling the *observed phenomena*. The resulting model aims to reproduce observed phenomena without necessarily being physically correct.

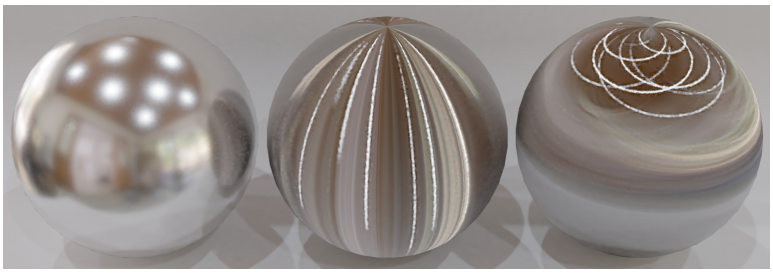


Figure 2.13: Left sphere: isotropic BRDF. Middle and right sphere: anisotropic BRDF. There exist large number of anisotropic reflections depending on microstructure direction.

- **Physically-based approaches** are building on certain physical foundations aiming to develop a physically plausible model.
- **Data-based approaches** are relying on actual real-world BRDF measurements.

Empirical approaches provide an intuitive understanding of light-surface interaction modeling in computer graphics through diffuse and specular components. Physically-based approaches are further building on those concepts as well as optics to deliver realistic light-surface interaction. Therefore, in this thesis, the physically-based surface reflection models are used.

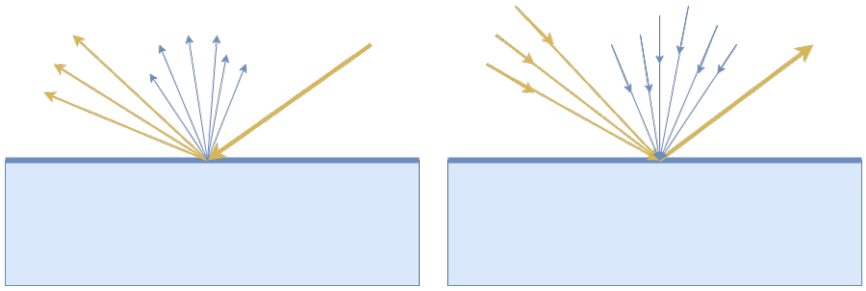


Figure 2.14: Two interpretations of BRDF. Left: given incoming light, BRDF specifies the distribution of reflected light. Right: given the view direction, BRDF specifies the relative contributions of incoming light. Image is recreated based on [30].

Empirical BRDF models

The most widely used empirical BRDF for *diffuse surface reflection* (Figure 2.15) is the *Lambertian model* [34]. Lambertian BRDF (Eq. 2.3) does not depend on incoming and outgoing direction ω_i, ω_o (e.g., light and view) and its only parameters are *albedo* also known as *diffuse color*. Albedo is either

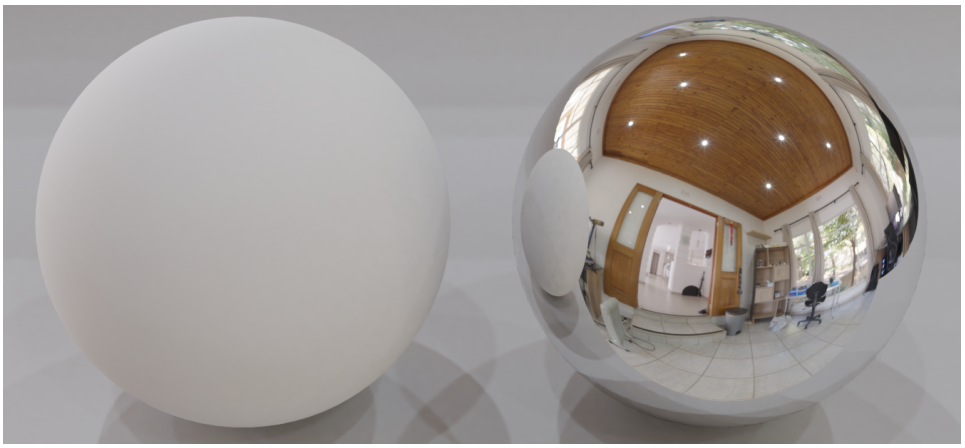


Figure 2.15: Left sphere: ideal diffuse, Lambertian reflection. Right sphere: ideal specular, mirror reflection.

a spectral or color (RGB) value. Lambertian BRDF can be interpreted as distributing reflected light equally in all directions. A wide range of opaque dielectrics can be represented with the Lambertian model.

$$f_{lambert}(\omega_i, \omega_o) = \frac{c_{albedo}}{\pi} \tag{2.3}$$

The simplest *specular reflection* model (Figure 2.15) is based on *mirror reflection* Eq. 2.4, which states that the angle of reflected light to surface normal is equal to the angle of incoming light to surface normal. It represents ideal reflection where incoming light ω_i is completely reflected into a single outgoing direction ω_o. Such model represents a perfect mirror surface which is completely sharply reflecting surrounding objects and lights. If the reflected incoming light direction from objects is not equal to the outgoing view direction, these objects will not be visible. Therefore, the specular reflection model depends on both incoming direction ω_i and outgoing direction ω_o around the surface normal n. Its only parameter is *reflectivity*. Reflectivity is a spectral or color (RGB) value and it determines the amount of light which is reflected in the outgoing direction. Since specular reflection represents light which is reflected from an object's surface it is present in both metal and dielectric materials as discussed by McDermott et al. [35]. In the case of dielectric material, reflectivity is equal to one and thus color of reflected light is equal to color of incoming light (e.g., light source). In the case of metal material, reflected light is tinted and thus reflectivity is represented with spectral or color (RGB) value.

$$f_{mirror}(\omega_i, \omega_o) = \begin{cases} reflectivity & if\ \omega_o == \omega_i - 2(\omega_i\ n)n \\ 0 & otherwise \end{cases} \tag{2.4}$$

Since many surfaces are rough and thus do not behave as perfect mirrors, a large body of research was devoted to improving the specular component. Early attempts were done by Phong [15] which inspired many more approaches. The concept of the Phong reflection model is based on the observation that many surfaces exhibit blurred reflections of light or other objects. This kind of blurred reflection is a result of *glossy surfaces*. Phong's idea was to represent a glossy surface as a broken mirror - a large collection of small mirror-like surfaces. For glossy surfaces represented this way, only a fraction of light rays are reflected into the eye direction causing the expected blurred reflection. These small mirrors were not modeled explicitly, but rather simulated using a clever trick. Phong observed that glossy reflection decreases as the angle between the view direction ω_o and ideal reflected direction ω_r increases (Eq. 2.5). Further, to control the amount of gloss, the dot product of outgoing (view) direction ω_o and reflected direction ω_r was set to the power of α. Smaller α results in a more rough, glossy surface, while higher α results in more mirror-like surface. As mentioned, the Phong model is empirical, meaning it represents observed phenomena without necessarily being physically plausible. Nevertheless, Phong's observations and model gave foundations for more complex empirical models such as Blinn-Phong [36] and Lafortune [37] as well as physically based models.

$$\begin{gathered} f_{phong}(\omega_i, \omega_o) = (\omega_o \cdot \omega_r)^\alpha \\ \omega_r = \omega_i - 2(\omega_i \cdot \ n)n \end{gathered} \tag{2.5}$$

Physically-based BRDF models

Modeling specular term represents quite a challenge for surfaces which are not ideal mirrors (i.e., not optically flat). Since most surfaces are not ideal mirrors and do exhibit glossy reflection, further research was done aiming for more physically based solutions.

State of the art models for specular reflection are based on the *microfacet theory* discussed by Torrance et al. [31]. Microfacet theory is based on the assumption that there exist surface variation (microgeometry) at scale smaller than the scale of observation (e.g., under a pixel) and yet greater than scale of visible wavelength, thus influencing light reflection. This surface variation is represented as a large number of microfacets - tiny, optically flat facets, behaving as perfect mirrors. Further, each microfacet has its normal m which slightly differs from the surface normal n. Only microfacets oriented so that incoming light is reflected in outgoing (view) direction would contribute. That is, only microfacets which are halfway between incoming and outgoing directions would contribute. Therefore, microfacet normal m must be equal to half-vector h to contribute. However, not all microfacets with such orientation whould contribute. Some microfacets are blocked by other microfacets not allowing incoming light to reach them, causing *shadowing*.

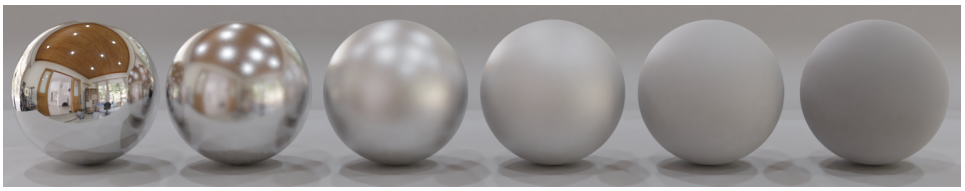


Figure 2.16: Microfacet-based BRDF with increasing roughness from left to right: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0.

On the other hand, certain microfacets can block the light reflected from other microfacets, not allowing light to leave the surface, causing *masking*.

Microfacet-based BRDF was introduced by Walter et al. [9]. Three main components for describing *microgeometry* are the arrangement of microgeometry normals, shadowing interactions and masking interactions. In microfacet theory, these components are not described explicitly, but rather using statistical distributions. To completely describe light interaction, Fresnel equations are used to specify the amount of light which is reflected. Therefore, microfacet-based specular BRDF (Equation 2.6) is described with following components: *distribution of microfacet normals* $D(\omega_h)$, *geometry term* $G(\omega_i, \omega_o, \omega_h)$ and *Fresnel reflectance* $F(\omega_i, \omega_h)$.

$$f(\omega_i, \omega_o) = \frac{F(\omega_i, \omega_h)G(\omega_i, \omega_o, \omega_h)D(\omega_h)}{4cos(\omega_i)cos(\omega_o)} \tag{2.6}$$

Distribution of microfacet normals (also known as normal distribution function NDF) is evaluated at half-vector h giving concentration of microfacets which *could* reflect light from incoming ω_i to outgoing ω_o direction. As discussed, shadowing and masking will prevent some microfacets from contributing. For most surfaces, microfacet normals m are pointing in the direction of surface normal n. NDF is a scalar-valued, often described with Gauss-like distribution. State-of-the-art NDF is GGX [9]. For isotropic surfaces, NDF is controlled with variance parameter often called *roughness*. Higher roughness implies a more irregular surface and thus a more blurred glossy reflection (Figure 2.16). Lower roughness implies more regular and thus sharp reflection. For anisotropic surfaces (Figure 2.13), two roughness values for tangent and bitangent surface direction are required.

Geometry term gives the percentage of microfacets with $m == h$ that are not shadowed or masked. Product of $D(h)$ and $G(\omega_i, \omega_o, h)$ gives the concentration of surface points that successfully reflect light from incoming ω_i to outgoing ω_o direction. Geometry term is crucial for energy conservation. Often, geometry term is represented using the Smith model [38].

Fresnel reflectance determines the amount of incoming light from direction ω_i which is reflected from each of the active microfacets. Besides incoming light direction ω_i, Fresnel reflectance depends on IOR which is

spectral or color (RGB) quantity. Often, Fresnel equations are replaced with Schlick approximation [39], [40] for more efficient computation.

It is worth noting that microfacet theory can also be used for transparent surfaces. In this case, microfacets are specularly transmissive [9]. Next, Oren et al. [41] used microfacet theory for a physically plausible description of diffuse surfaces. Further, multiple scattering between microfacets is an additional improvement that has to be considered [42]. Additionally, microfacet models can be extended to take into account wave optics such as diffraction or interference as discussed by Steinberg [43]. Finally, the micro-geometry model is relatively limited and builds on assumptions which are not always true, e.g., the assumption that micro-geometry normal and height are uncorrelated. Enhancements on microfacet geometry are discussed by d'Eon et al. [44].

Microfacet-based BRDFs are a state of the art approach for modeling metal surfaces which exhibit specular and glossy reflection and thus are used in this work. Microfacet BRDFs well describe both color and directional material characteristics (e.g., glossy reflection from the smooth or rough surface). However, describing surfaces with constant microfacet BRDF parameters, for each surface point, results in homogeneous, uniform, too-perfect and thus not realistic synthetic surfaces. The crucial element for synthesizing realistic surfaces is modeling surface complexity through *variation* of color, directional properties and small-scale details. The aim is to reproduce surface patterns (e.g., surface finishing of machined surfaces), surface non-uniformity (e.g., variation in color or roughness) and small-scale geometrical imperfections (e.g., small-scale bumps, dents and scratches). This problem was also highlighted by Jakub et al. [45] and Yan et al. [46] who extended microfacet-based BRDFs so that surfaces can exhibit high-frequency surface details. That said, modeling surface texture is crucial for the synthesis of realistic surfaces and it will be the focus of this thesis.

Data-based BRDF models

The foundation of data-based BRDFs is surface reflection information measured from real-world surfaces. BRDF measurements are often stored as look-up-tables and parameterized using incoming and outgoing directions as discussed by [47]. Alternatively, BRDF data can be used for creating *neural materials* [48] which can be seen as compression of BRDF measurements.

Besides measuring and storing pure BRDF data, e.g., material directionality effects and color without spatial variation, Dana et al. [49] introduced measurement and storage of both surface reflectance and texture as *bidirectional texture function* (BTF). Further, measured reflectance and texture can be used for creting neural materials [50] which can again be seen as the model for compression and efficient query.

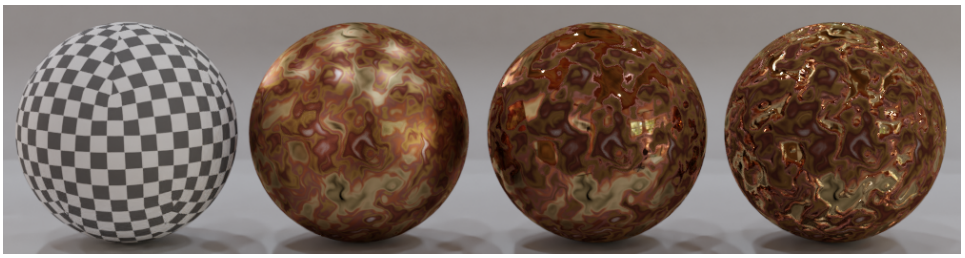


Figure 2.17: Surface modeling in computer graphics is performed on different scales. For the first sphere, only geometry modeling and thus modeling of macroscale is performed. For the second and third sphere, besides geometry, color and roughness are modeled, therefore both macro- and microscale modeling is performed. In the fourth sphere besides geometry, color and roughness, variations of surface normal is modeled representing micro-, meso- and macroscale modeling.

Data-based BRDFs exhibit impressive results and often serve for evaluation of analytical BRDF models or realistic recreation of existing real-world surfaces as discussed by Ngan et al. [47]. However, data-based BRDFs hold several downsides. The most obvious is the memory requirement for storing the measurements for dense pairs of incoming and outgoing directions. Further, they are not controllable since there are no parameters directing their behaviour.

2.3.3 Texture Modeling

The texture does not have a well-established definition due to the complexity and rich diversity of existing, real-world surface textures. In computer graphics, for material modeling, texture can be seen as a function which assigns each point of an object surface a certain value. Textures are used to introduce the variation of surface properties such as color, directional effects or small-scale surface geometry.

Texture can appear on a multitude of scales, therefore, it is needed to introduce three main spatial scales on which modeling in computer graphics is performed (Figure 2.17). It is important to note that the scale of modeling is determined by viewing distance. *Macro-scale* is represented with object geometry and describes its shape as discussed in Section 2.2. *Meso-scale* represents spatial variations visible by the eye (e.g., spanning over several pixels) but are smaller than object geometry. Meso-scale surface details modeling is discussed further in this section. *Micro-scale* is represented with BRDF and describes microgeometry not directly visible by the eye, in other words, under a pixel as discussed in Section 2.3.

Textures are used to modulate BRDF parameters (e.g., reflectance or roughness) or to add *mesoscale* surface details (e.g., perturbing surface nor-

mals) as it can be seen in Figure 2.17. Coupling textures with BRDF is used to introduce surface variation, eliminating uniform and overly perfect surfaces, which is crucial for realistic surface modeling and thus image synthesis. It is important to note that texture modeling in computer graphics serves as an efficient method of adding surface details to object shapes. Therefore, a large body of work in computer graphics was devoted to texturing: creating texture models and applying them for material modeling for realistic image synthesis. Texturing methods can be roughly separated into *image texturing* and *procedural texturing*.

Image Texturing

The image-based texturing approaches uses image data which carries information about BRDF parameters and/or mesoscale geometry. That is, each surface point is mapped to a certain pixel of image texture (also called *texel*).

Image textures can be generated by painting, scanning, photographing, photogrammetry ([51], [23]), etc. Textures can be drawn directly on a 3D surface and stored as a collection of images [52]. Alternatively, textures can be created using algorithmic pattern description, for example, noise [53]. Image textures can be further created from complex 3D geometry and materials using the process called *baking* [54]. Further, image data can be created using generative AI or exemplar-based approaches. Generative AI is often based on generative adversarial networks (GANs) as discussed by [55]. Exemplar-based texture synthesis creates a new texture based on an input sample or exemplar (e.g., an image) as discussed by Wei et al. [56]. In the first step, analysis is performed to find features of the exemplars of pixel level [57] or patch level [58]. In the second step, the texture is synthesized using found features and a set of rules.

During rendering, image textures are accessed with 2D coordinates (e.g., (s, t)). Each pixel of a 2D image contains certain surface properties such as BRDF parameters or mesoscale normal. For the sake of completeness, it is worth mentioning, that 3D textures are direct extensions of 2D textures and accessed with 3D coordinates (e.g., (s, t, r)). 3D textures are useful for describing volumetric objects since they carry certain volume properties such as density which can be evaluated during rendering as discussed by Wilson et al. [59]. For this thesis, the focus will be on 2D image textures.

Once 2D image texture is generated, to be used in the shading process of the rendering procedure, it has to be mapped to a 3D object geometry, which leads to phrase *texture mapping*. Texture mapping can be seen as a problem of mapping from the 2D domain to the 3D domain (or vice-versa). The process of texture mapping and using its values for rendering can be described using *texturing pipeline* (Figure 2.18) discussed by Akenine et al. [1], ch. 6.1., and which will be now shortly presented.

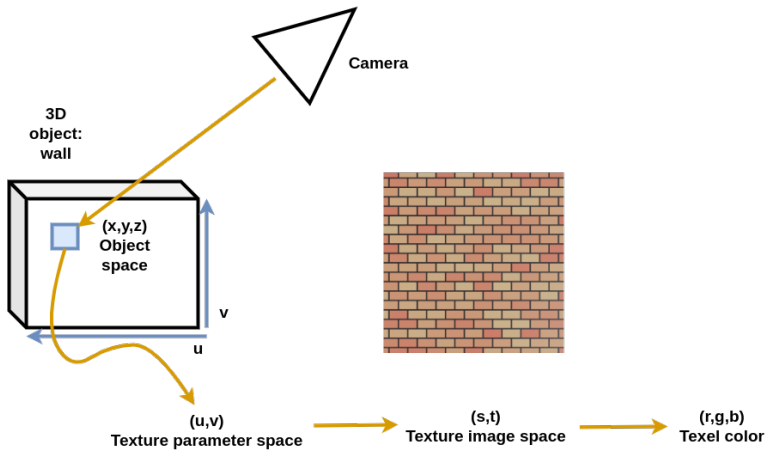


Figure 2.18: Texturing pipeline recreated as discussed by Akenine et al. [1].

Input to the texturing pipeline is *object space* location - a 3D coordinate (x, y, z). This value is calculated by rendering procedure while resolving which objects are visible from the camera. Object space location (x, y, z) is then used as an input to *projector function* which maps those coordinates to 2D *texture parameter-space* coordinates (u, v). The projection function can be implemented in many ways. One common approach is interpolation of texture coordinates which are precalculated using mesh unwrapping. Mesh unwrapping [60] poses a complex problem for objects of complex geometry and thus certain extent of experience and manual involvement is required. Another common approach is to employ texture projection functions which project texture from simple shapes such as planes, spheres, cylinders and cubes as discussed by Pharr et al. [14], ch. 10. Texture projection functions can be applied to complex geometry by first separating the geometry into simpler shapes and then using appropriate projection based on the shape. Once 2D texture coordinates (u, v) are obtained, *corresponder function* is used to map those coordinates to *texture image-space* location (s, t) which has the range of image height and image width. The corresponder function can further perform transformations of texture (i.e., scaling, rotating and translating). If the texture is smaller than the size of the surface, then the corresponder function is responsible for texture wrapping: repeating, mirroring, clamping, etc. Once texture image-space location (s, t) is available, the value from that specific image texture *texel* can be obtained. Those are usually RGB or floating values. Finally, obtained texture values are transformed to the type required for evaluating the BRDF. This is needed since obtained image texture values can be interpreted differently than the original value. For example, the obtained RGB value can be interpreted as a vector and thus RGB image as a vector array.

Two important concepts which must be discussed when using image textures are *texture magnificiation* and *texture minification* as discussed by Akenine et al. [1], ch. 6. When image texture is magnified to the extent where one image texture texel covers several pixels of the final image, then a pixelized, stairs-like effect is visible. To solve this, filtering, such as nearest-neighbour, bilinear, cubic, etc. is used for smoothing and reducing the blocky effect. On the other hand, if multiple image texture texels fall under one pixel of the final image, then the aliasing might occur resulting in artefacts known as *Moire pattern*. The problem of aliasing can be explained with sampling theory and solved if the Nyquist sampling theorem is respected. In the case of 2D images, this means either to increase sampling frequency per pixel or to decrease the frequency of texels influencing one pixel. Increasing sampling frequency (Figure 2.20) by using *multiple samples per pixel* is often used as well as in this work (Section 2.6). To decrease the frequency of texels, *mip-mapping* is used in which the original image texture is downscaled and filtered to create *image pyramid* [61]. Smaller versions of image texture are used when there is a larger distance between the camera and the texture surface. On the other hand, larger versions of image texture are used when there is a smaller distance between the camera and the textured surface. This way, the required one-to-on pixel-to-texel ratio can be achieved as discussed by Akenine et al. [1], ch. 5.

Procedural Texturing

The procedural texturing approach generates BRDF parameters and/or mesoscale surface information using an algorithm. More specifically, each point on an object, including any additional information in this point such as surface normal, is evaluated in a procedure resulting in a value that can be used for modulating surface properties.

Procedural texturing is an umbrella term for a number of techniques in computer graphics to create textures from a set of rules - procedures and algorithms that specify certain patterns as discussed by Ebert et al. [62] and Vivo et al. [63]. Therefore, instead of image lookup as done for image textures, procedural textures are evaluated from a function. Thus, procedural texture can directly be seen as a function which defines values for each point in 3D space or 3D surface.

Developing a procedural texture often requires creating and combining regular and irregular patterns (Figure 2.19). Regular patterns are often based on mathematical analytical expressions such as circles, lines, sine, etc. Although very efficient, regular patterns alone are too ideal for representing the complexity of a realistic surface. Therefore, irregular patterns which break regularity and ideal patterns are used.

Irregular patterns are often based on noise functions. Noise functions are often combined to achieve more complex patterns. Examples of such

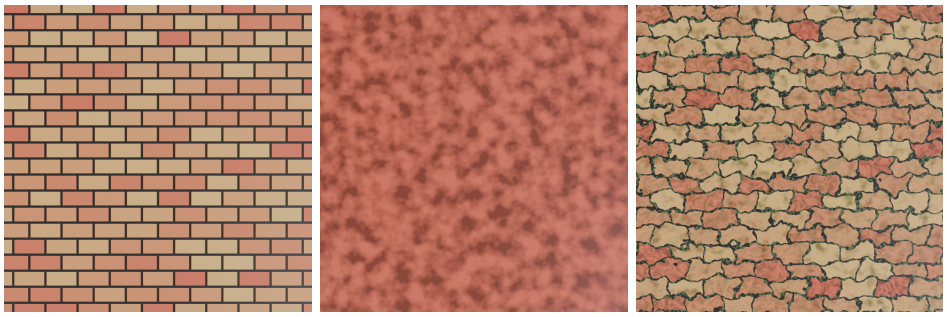


Figure 2.19: Left: regular pattern. Middle: irregular pattern. Right: combination of regular and irregular patterns.

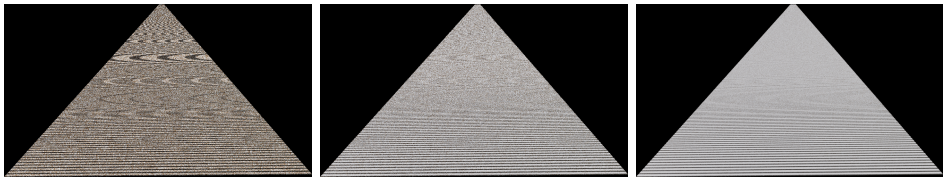


Figure 2.20: Simple procedural texture depicting stripe pattern. Images left, middle and right are rendered with 1, 4 and 32 samples per pixel respectively. Note how the image with a single sample per pixel exhibits a strong aliasing effect which is reduced using multiple samples per pixel.

building blocks are Perlin noise [11] and Worley noise [64]. Although irregular, noise functions are controllable and hold favourable properties such as gradual changes in local values, and larger changes in global values. These methods very well represent irregular structures from micro to macro size in nature and thus are often used for modeling natural phenomena and can be extended for modeling defects on manufactured surfaces as discussed by Dorsey et al. [28].

In this work, procedural texturing methods are used for describing product surfaces. Such surfaces are created using certain manufacturing method which requires both regular and irregular patterns to be combined. Procedural texture modeling is crucial for this work and will be further discussed in Chapter 3.

Similarly as for image textures, using high-frequency procedural textures often leads to aliasing. One solution is to remove high-frequency information from procedural texture before sampling it as discussed by Pharr et al. [14], ch. 10. Another solution is to use multiple samples (Figure 2.20) per pixel during rendering which is also used in this work.

Figure 2.21: Texture used for different amounts of surface geometry variation. Left: bump/normal mapping. Middle: displacement mapping. Right: displacement mapping with higher intensity. Normal mapping or bump mapping can serve as a good approximation for displacement mapping when geometrical features are relatively small compared to the whole 3D object geometry. Note how displacement mapping moves actual geometry exhibiting irregular silhouettes and shadowing

Texture and material modeling

Image and procedural textures are used to modulate BRDF parameters or to introduce mesoscale surface details for achieving realistic surfaces. BRDF parameters which are often modulated are albedo, reflectance and roughness (Figure 2.17). Meso-scale surface details are introduced by perturbing surface normals using height maps or normal maps. Perturbed surface normals are then used during BRDF evaluation. This way various surface details can be represented such as surface finishing patterns, bumps, dents, scratches, etc.

For this work, textures are especially interesting for introducing small-scale surface details. An example is modeling a realistic machined product surface, which requires modeling both surface finishing patterns and surface imperfections. These mesoscale geometrical details are often introduced using *bump mapping*, *normal mapping*, *parallax mapping* or *displacement mapping* (Figure 2.21).

Bump mapping is discussed by Mikkelsen [65]. Input to this approach is the desired surface height field which is given either by procedural or image texture. Height value is used to perturb surface normals. Once BRDF is evaluated on a surface containing perturbed normals, the surface *appears* to be containing small-scale details. It is important to note that this method does not change actual surface geometry. It only perturbs the normals of a surface. Thus it is a very efficient method for introducing surface details, but it also exhibits significant drawbacks. First, details described by bump mapping can not cast a shadow. Second, described details are not occluding other details making the surface look flat if seen from a grazing angle.

Normal mapping is similar to bump mapping, but instead of a height

map, a normal map is used as input. Normal map encodes perturbed normals directly as pixel colors. Normal mapping is also an efficient method for introducing details but holds the same drawbacks as bump mapping. However, normal mapping is a widely used method for an efficient description of mesoscale surface geometry, and thus improvements such as solving shadowing and masking of light [66] are introduced.

Parallax mapping [67] uses a height map to achieve a similar effect as bump mapping but also takes into account the viewing position causing parallax and occlusion of surface details. However, this method has a higher computational cost and still is more of a trick than an actual solution since no actual geometry is generated.

Displacement mapping [68] is the most computationally demanding method. It requires a finely subdivided mesh and height map. This method moves vertices of mesh in the direction of the surface normal by the amount specified by the height map. Therefore, this method changes actual surface geometrical details enabling self-shadowing, occlusion and parallax eliminating problems with bump, normal and parallax mapping.

2.4 Light

In the real world, without light and light sources, it would not be possible to see anything. Further, the shapes and properties of real-world light sources are various, highly influencing how we observe the world around us. As realistic image synthesis aims to reproduce imagery which looks as it is observed in the real world, it is crucial to model light sources in 3D scenes so they represent real-world light sources as closely as possible.

In computer graphics, light modeling is often based on geometric optics which assumes that light travels along rays from light sources and between objects in the 3D scene. Only a certain small amount of light enters the camera forming the image. The flow of light through the 3D scene is computed during rendering, more specifically, during shading and light transport (Section 2.6). In this section, a short overview of light modeling in computer graphics is provided.

2.4.1 Characterization of Light

Radiometric techniques in optics describe light as electromagnetic waves in the visible spectrum with wavelengths ranging from 380nm to 780nm. Light emitted from a light source is characterized by *spectral power distribution* (SPD) as discussed by Pharr et al. [14], ch 5. SPD is the distribution function of wavelength which describes the amount of light at each wavelength. Light falling on an object's surface will absorb and reflect. The reflected light is perceived as color of the object surface (Figure 2.22). The amount of light reflected by object surface is described with *spectral reflectance curves* as

Radiometric measurements and their photometric analogs			
Radiometric	Unit	Photometric	Unit
Radiant energy	Joule (Q)	Luminous energy	Talbot (T)
Radiant flux	Watt (W)	Luminous flux	Lumen (lm)
Intensity	W/sr	Luminous intensity	lm/sr=candela(cd)
Irradiance	W/m^2	Illuminance	lm/m^2=lux(lx)
Radiance	$W/(m^2sr)$	Luminance	lm/(m^2sr)=nit

Table 2.1: Comparison of radiometric and photometric measurements.

discussed by Pharr et al. [14], ch 5. Three disciplines dealing with light characterization are:

- **Radiometry** deals with the measurement of light radiation and physical transmission of light. Fundamental radiometric quantities are energy, flux, intensity, irradiance and radiance (Table 2.1).
- **Photometry** is similar to radiometry except that it takes the human visual system into account. Therefore, each radiometric quantity has an equivalent photometric quantity: energy, flux, intensity, illuminance and luminance (Table 2.1). Conversion of radiometric quantities to photometric ones is done using *CIE photopic spectral luminous curve* which describes the average sensitivity of the human eye to different wavelengths for well-lit conditions.
- **Colorimetry** deals with the perception of color and its relationship to SPD and thus defines *color spaces* and *gamut*.

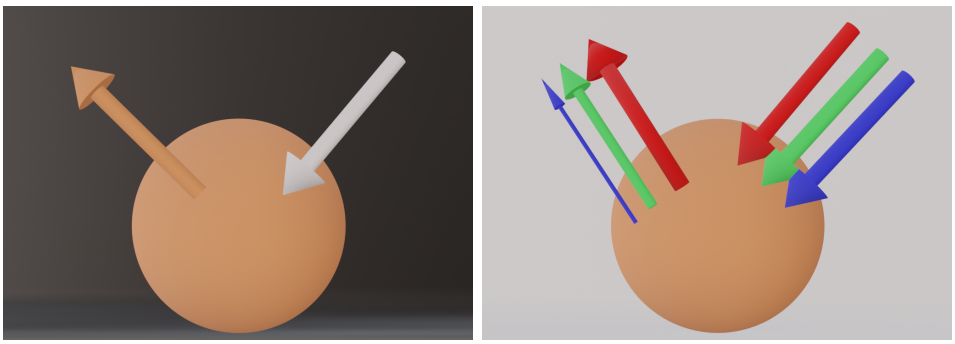


Figure 2.22: Light falling on an object will absorb and reflect. The reflected light is perceived as color of the object. Left: white light is absorbed and only orange color is reflected. Right: in terms of RGB, white light contains an equal amount of RGB while after absorption, red and green are mostly reflected which is observed as orange.

2.4.2 Color

Different wavelengths of light are perceived as color by the human visual system. Thus, color is a perceptual (psychophysical) phenomenon rather than a physical one. The crucial element of the human visual system for color perception is *photo-receptive cells* - receptors at the back of the eye (i.e., retina) as discussed by Pharr et al. [14], ch 5. The two main cell types are cone and rod cells. *Cone cells* are active in well-lit environments (e.g., daylight). They are sensitive to red, green and blue (R, G, B) light. Thus they serve for the transformation of light radiation into (R, G, B) color - *trichromatic color vision*. On the other hand, rod cells are sensitive to smaller amounts of light and thus active in low-lit environments. Further, rod cells are not reproducing colors.

Color is characterized by *color space* and *gamut*. *Color space* is a model representing colors perceived by the human visual system. It gives a numerical representation of color which can be used for 3D scene description as well as computation in rendering. *Gamut* describes the range of possible color that can be represented by a particular color space as discussed by Smith [69].

For modeling and rendering purposes, in most cases specifying light as SPD and achieving wave effects such as polarization and diffraction is not needed. Therefore, it is more practical to represent light as color. Trichromatic color vision motivated *XYZ color space* as a bridging color space needed for converting light characterized by SPD into color. XYZ color space can represent any color visible to the human visual system. Conversion of SPD of light into XYZ color space is done using *CIE standard observer color matching functions* which were constructed based on *color matching experiment* [14], ch. 5. Once the SPD of light is converted into XYZ color space it has to be further converted into, e.g., *RGB color space* to be visualized on a display device.

RGB color space is the most often used color space for rendering and display. RGB color space can be visualized as a unit cube where each point represents one color. Since both RGB and XYZ are color spaces, then linear transformation, described by 3x3 RGB-to-XYZ matrix, is used for conversion [14], ch. 5. It is worth mentioning that RGB color space is *additive* color space, as opposed to *subtractive* color space. New colors in an additive color system are made by adding two or more colors together. For example, light sources of different colors are added into a new color. On the other hand, new colors in subtractive color system are made by subtracting two or more colors. For example, color of light reflected from a surface is changed due to absorption which can be seen as subtracting.

2.4.3 Light Sources

This work relies on geometric optics where the light is represented with rays and the behaviour of light with matter is described with IOR. Therefore, the description of light using photons or waves is out of scope. This further implies that effects such as interference, polarization, diffraction, etc. will not be included. That said, light sources must, at least, define the position or direction, color and intensity of emitted light. Additionally, light sources can have a certain size and shape. Such light sources are called *physical lights* (Figure 2.23). Lights without size and shape are called *non-physical* lights (Figure 2.23).

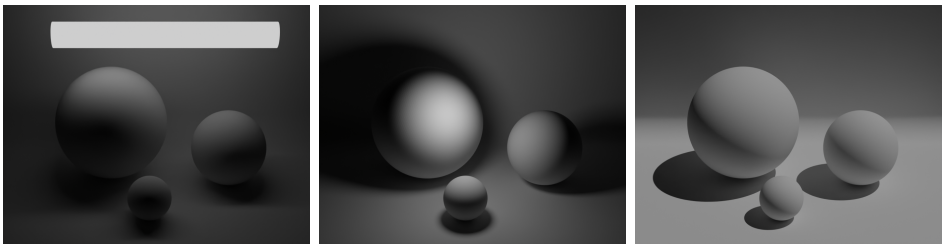


Figure 2.23: Left: physical light source with cylinder as emissive geometry. Middle: point light source. Right: directional light source. Point and directional light sources are examples of non-physical light sources which do not have size or shape.

Physical Lights

Physical lights have certain size and shape, described with geometry as discussed by Pharr et al. [14], ch. 12. The geometry of physical light can additionally have certain transformations, e.g., position and orientation in a 3D scene. The emission property of each surface point is defined with *emissive distribution*. Since real-world light sources have certain shapes and emission distributions, physical light sources serve as the best approximation. It is important to note that rendering of 3D scenes containing physical lights requires more complex light transport methods (Figure 2.24) as well as geometry sampling which will be discussed in Section 2.6.

Non-physical Lights

Non-physical lights serve as approximations of physical lights as discussed by Pharr et al. [14], ch. 12.. They do not have size or shape and thus the name *non-physical*. However, they give information on light position, direction, intensity and color. Using this information, it is possible to approximate the behaviour of the light sources, namely light falloff with distance, which

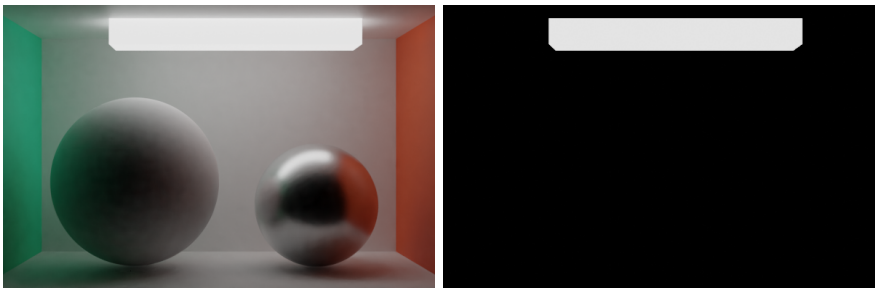


Figure 2.24: Same scene rendered using ray-tracing-based rendering (left) and rasterization-based rendering (right). Rendering illumination coming from physical lights requires advanced light transport discussed in Section 2.6.

objects are illuminated, which objects cast shadows, etc. Two main types of non-physical lights are *directional lights* and *point lights*.

Directional lights (also known as parallel or distant lights) simulate light sources which are considered so far from objects in a 3D scene so that their emission can be represented by parallel rays. Therefore, directional lights are completely characterized by their direction, color and intensity. Note that light emitted from directional lights is constant over a 3D scene (i.e., no light fall-off) and that the position of directional lights does not influence its illumination properties.

Point lights (also known as spherical, omnidirectional or punctual lights) are infinitesimally small in size and thus represented by a point. Light from point lights is emitted in all directions uniformly. Therefore, point lights are characterized by their position in the 3D scene, color and intensity. The position of the point light determines the distance and direction of incoming light for each surface point of objects in 3D scene. Distance of point light from a surface can be used to compute light falloff (Equation 2.7).

$$L_{point} = (light\_intensity)(light\_color) f_{distance}(r) f_{direction}(l) \tag{2.7}$$

Inverse square law falloff (Equation 2.8) is most often used. Since light from point light is redistributed over a sphere, as the sphere gets larger, energy gets redistributed over a larger area of the sphere. Therefore, more distant objects receive less light. For practical usage, small value ϵ is added to distance r since the intensity of light becomes close to infinite if the distance of lights to objects is close to zero as discussed by Akenine et al. [1], ch. 5.

$$f_{distance}(r) = f_{inverse\_square\_falloff}(r) = \frac{1}{4\pi(r+\epsilon)^2} \tag{2.8}$$

Besides distance fall-off which is a type of light variation by distance, directional fall-off can be used to describe light variation by direction. Directional

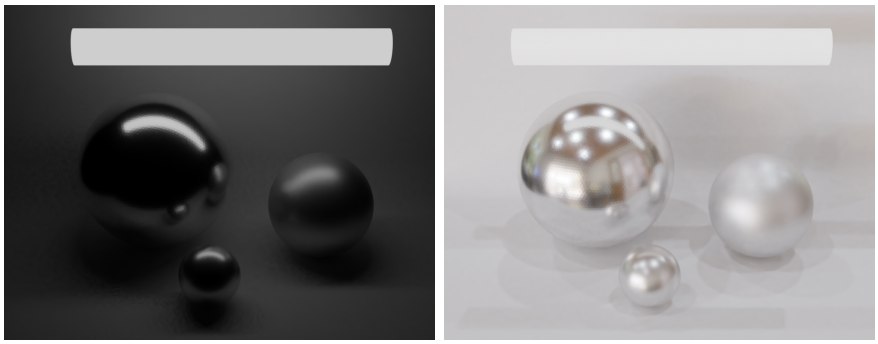


Figure 2.25: Left: only physical light source with the cylinder as emissive geometry. Right: environment light and physical light source. Note how environment light is used to approximate large light sources emitting light from all directions.

variation enables the creation of a wide range of different light sources. Most often variations of point lights are *spotlights* which have cone-shaped directional light falloff as discussed by [14], ch. 12. Combining distance and directional light fall-offs with a variation of intensity and color using textures, point lights can be used to derive a wide range of rich light sources as discussed by [14], ch. 12. More precise variation for light distribution is represented using Illuminating Engineering Society (IES) profiles [70]. IES profiles enable simulating distributions of real light sources by measuring their emission over angle.

Environment Light

To approximate light coming from large sources of light such as skies, environment light is used (Figure 2.25). In this case, variation in light intensity and color depends only on incoming light direction and not position. Therefore, environment light can be described with *spherical functions* defined over unit sphere directions. Spherical functions can be represented using tabulated forms, spherical basis and environment mapping. The most widely used method for representing environment illumination is *environment mapping* as discussed by Greene [71]. In this case, environment light is stored in an (image) texture called environment map. The simplest type of environment mapping is reflection mapping where the reflected vector from the view vector is used to read the environment map light value. For this purpose, latitude-longitude, sphere mapping or cube mapping can be used.

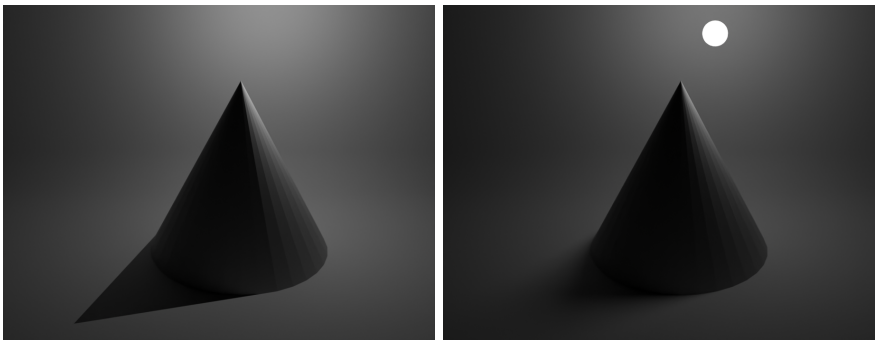


Figure 2.26: Left: point light source has infinitely small size and thus only enables casting hard shadow. Right: physical light source with sphere geometry casts soft shadow: umbra and penumbra component.

2.4.4 Shadows

Shadows are a crucial visual cue for understanding object placement and relationship with other objects in a 3D scene (Figure 2.26). Further, shadows are crucial for the visualization of small surface details (Figure 2.21). Light coming from the light source is occluded by 3D objects called *occluders* which cast a shadow on other 3D objects which are called *receivers*. Cast shadow can be decomposed in *umbra* - dark shadow and *penumbra* - soft shadow around umbra (Figure 2.26). Small light sources such as point lights cast hard shadows which contain only umbra component. Area light sources such as physical lights cast soft shadows containing both umbra and penumbra component. The core component required for computing shadows in 3D scene is solving visibility between light sources and surfaces as well as between surfaces. Visibility solving and computation of shadows will be further discussed in Section 2.6.

2.5 Camera

The camera model in computer graphics is based on real-world camera concepts for realistic simulation. Therefore, we will shortly review the main components of the *real-world camera system*. Camera components are placed in *light-proof enclosure*. Only by opening *shutter*, light can enter the camera. The light which enters the camera travels through *lens* system which focuses light through a small opening called *aperture* and falls on a digital *sensor*. Light falling on the digital sensor is transformed into color resulting in an image. The time in which the sensor is exposed to the light is called *exposure time*. Exposure determines image brightness and the amount of details visible in dark or light areas. Exposure is controlled with *aperture size*, *shutter speed* and *sensitivity (ISO) value*. A larger aperture enables

more incoming light but also smaller *depth of field* (DOF) - the distance between the nearest and furthest object from the scene that appears sharp in the formed image. A longer shutter speed enables more light to fall on the digital sensor while also causing *motion blur* for moving objects. Finally, larger sensitivity enables more exposed images but with potential noise in darker scenes.

Pinhole camera represents the simplest camera model. It is a light-proof box which contains only one small opening - *aperture* and light-sensitive *film* on the opposite side. Objects in the real world reflect light and some of this light enters the aperture, falling on the film. Image formation can be explained by following the ray connecting objects outside of the camera through the aperture on the film. The image formed on the film using a pinhole camera is flipped. In the real world, the aperture has a certain, finite size and thus each point in the visible scene can map to multiple points on the film causing blur. A larger aperture enables more light to pass to the film but the image is blurred. A smaller aperture results in sharp images but enables a lower amount of light to pass. For this reason, a *lens* is placed in front of the aperture to enable a larger aperture opening while evading blur by gathering (converging) the light.

An *ideal pinhole camera* has an infinitely small aperture where each point in a visible scene maps to exactly one point on the film. The main parameters of the ideal pinhole camera are the distance of the film to the aperture and the size of the film. Distance of aperture to film is called *focal length* or *focal distance*. A larger focal length effectively results in zooming in, while a smaller focal length effectively results in zooming out. Further, a larger focal length enables smaller *angle (field) of view* and vice versa. Finally, a larger *film size* implies a smaller angle of view and vice versa.

Virtual pinhole camera is the fundamental camera model in computer graphics, inspired by the ideal pinhole camera, as discussed by Pharr et al. [14] ch. 6. In the case of a virtual pinhole camera, the aperture is placed in front of the film for more tractable computations. Although this kind of setup is not possible in the real world, it respects all the rules of geometric optics and thus it gives the same result as a pinhole camera, except that the resulting image is not flipped. The virtual pinhole camera is thus defined with *aperture* - a three-dimensional point and *film (image) plane* with a certain distance from the aperture. The main parameters are *focal length* (Figure 2.27), *film (image) dimension* (Figure 2.28) and *resolution* (Figure 2.29). It is important to note that different resolutions can be used for the same film dimension and vice versa. Film dimensions determine the angle of view while resolution only determines the quality of the image (e.g., amount of details) and not the angle of view. Further, the camera has transformation parameters (Figure 2.30), e.g., position and orientation in a 3D scene. Discussed camera properties and transformations determine the portion of the visible 3D scene. Virtual pinhole camera further defines

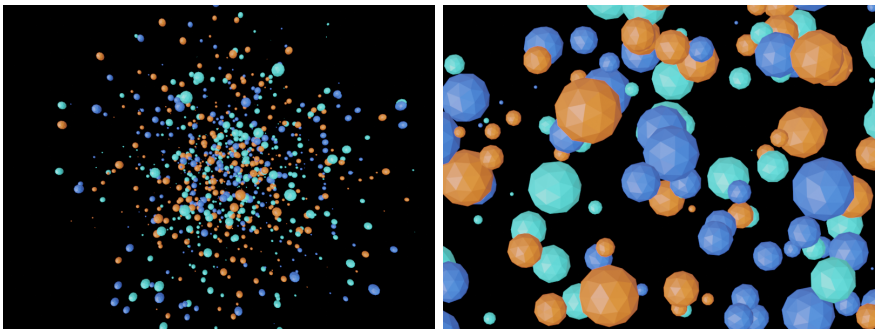


Figure 2.27: All camera parameters fixed except focal length. Left: focal length of 180mm. Right: focal length of 20mm. Note how focal length effectively means zooming out (left) or zooming in (right).

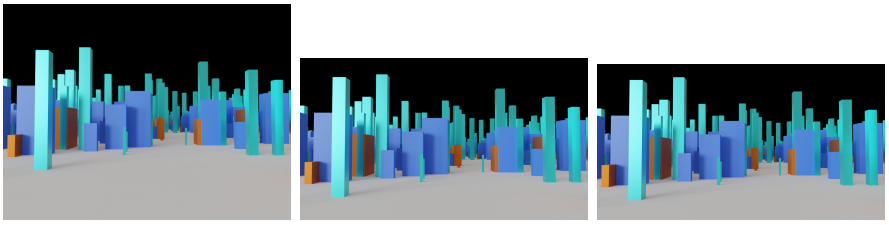


Figure 2.28: Various film dimensions (aspect ratios). Left: 4:3. Middle: 16:9 Right: 1.85:1. Note how different portion of the scene is visible although all other camera parameters are fixed.

near and far clipping planes (Figure 2.31) which are used for clipping very close and very far objects in 3D scene to ensure numerical stability during rendering.

Two important types of virtual cameras are perspective and orthographic (Figure 2.32). *Perspective camera* simulates the *foreshortening effect*. Foreshortening is present in the human visual system and simulated by real camera systems, where more distant objects appear smaller. Therefore, it is a prerequisite to achieving photo-realistic image synthesis. An *orthographic camera*, on the other hand, generates images of objects which have the same size regardless of the distance. As such, are quite useful for 3D modeling purposes.

The virtual pinhole camera is used for image formation during rendering. We will shortly discuss how pinhole camera is used for two practical rendering approaches: ray-tracing-based rendering and rasterization-based rendering which are more discussed in section 2.6. In *ray-tracing-based rendering*, the image is formed by creating a ray which connects the aperture point and each pixel of the image plane. This ray is called *camera ray* and it is traced into a 3D scene. If the intersection of the camera ray and object is found, then the corresponding point of the pixel on the image plane will

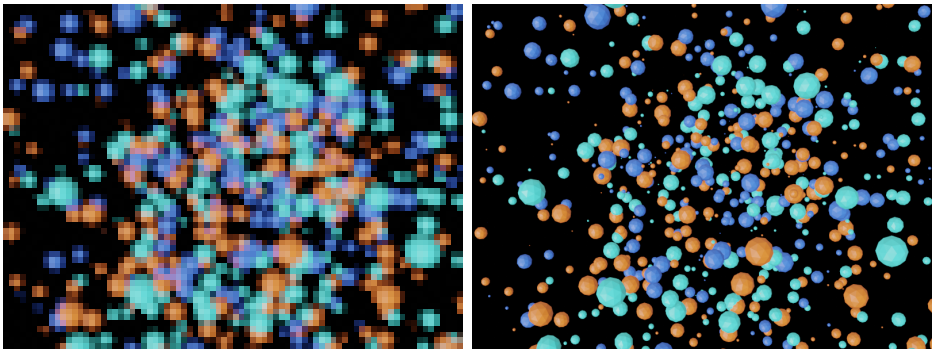


Figure 2.29: All camera parameters fixed except resolution. Same film dimensions (aspect ratio 4:3). Left: 80x60 pixels. Right: 800x600 pixels. Note how the angle of view (portion of the scene) is the same and only quality differs.

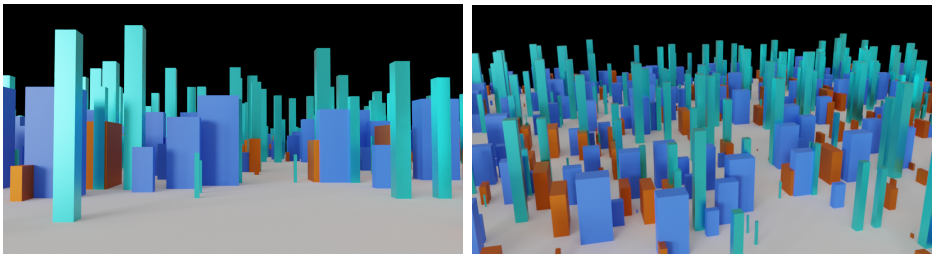


Figure 2.30: All camera parameters fixed except transformation (position and rotation). Note how visible portion of 3D scene depends on camera transformation parameters.

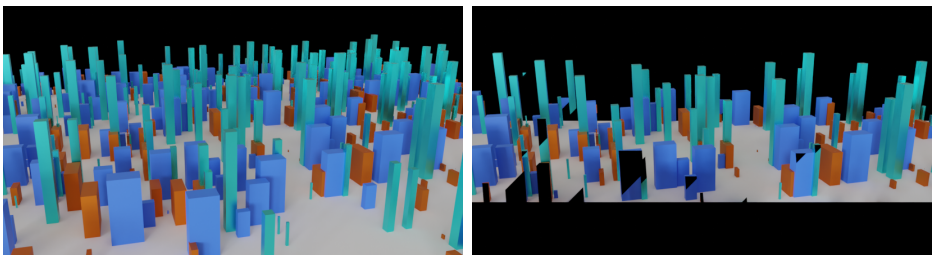


Figure 2.31: Left: whole 3D scene visible from camera's point of view. Right: 3D scene visible from the camera's point of view with near and far clipping planes moved closer.

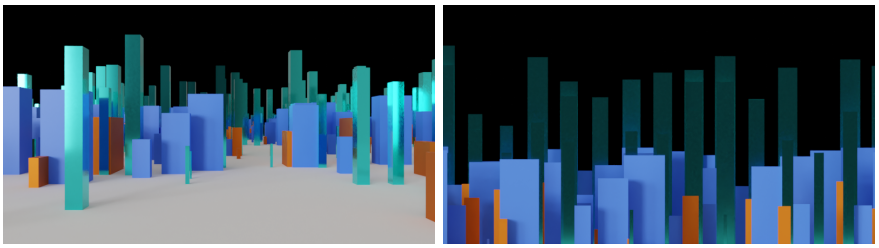


Figure 2.32: Left: perspective camera. Right: orthographic camera.

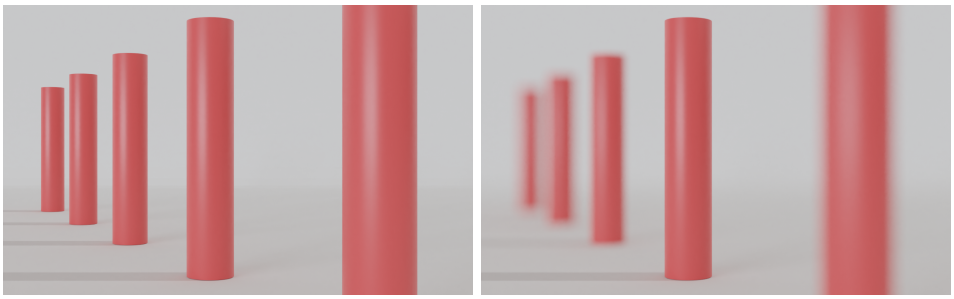


Figure 2.33: Left: rendered image without DOF. Right: rendered image with DOF.

be assigned with the color of the intersected surface point. In *rasterization-based rendering*, the camera is represented using a perspective projection matrix which contains the camera properties. This matrix projects triangle vertices of mesh objects onto the image plane. Projected triangles are then used for determining the color of image plane pixels.

The main problem with the described virtual pinhole camera is that all objects are in sharp focus due to infinitely small aperture which is not possible in the real world. As discussed, real-world cameras use lenses, while pinhole cameras neglect the important effects that the lens system has on light passing through it. The most obvious downside of the virtual pinhole camera is that DOF is not simulated (Figure 2.33). Furthermore, the amount of light entering the lens system and leaving it is different. Thus, lenses introduce various *abberations* such as vignetting which causes a darkening toward the edges of the image. Also, lenses introduce distortions such as barrel distortion where straight lines are imaged as curves. Therefore, lens simulation presents a crucial part of realistic image synthesis. The simplest model of lens camera is *thin lens* camera discussed by Heidrich et al. [72] which contains a finite aperture opening and spherical lens without thickness. An improvement is *thick lens* camera discussed by Kolb et al. [73] which approximates finite aperture opening with a lens of finite thickness. Finally, a model of a lens system containing a number of lenses is modeled for achieving a realistic camera as discussed by Wu et al. [74].

2.6 Rendering

Rendering is a process of generating a 2D image from a 3D scene. Thus, to view a 3D scene containing 3D objects and lights from a particular camera position, rendering must be performed (Figure 1.6). The generated image is intended for the raster display device, therefore, the rendering process generates raster images. A raster image is a two-dimensional array of pixels (discrete picture elements). Each pixel stores discrete color intensity. Therefore, rendering can be seen as a discretization of a 3D scene.

The rendering process can be intuitively understood by observing what happens when a photograph is taken in the real world. First, the camera is positioned and oriented to capture the object of interest. Secondly, the camera shutter is opened enabling light to enter the camera. The light entering the camera is only the small amount of light which was first emitted from a light source (e.g., the Sun) and scattered across the scene surrounding the desired object. Therefore, light entering the camera depends on the positions and properties of light sources and objects. The light which enters the camera is transformed into an image. The created image contains the desired object and portion of the scene visible to the camera.

Therefore, to simulate the process of image generation it is required to simulate the amount of light emitted from a light source, scattered across the scene and entering the camera. This approach to light simulation is called *forward tracing*. Since only a small amount of computed light is entering the camera, it is extremely inefficient to simulate all light emitted from the light source and scattered across the 3D scene. Therefore, rendering simulates only the light entering the camera. This approach is called *backward tracing*.

Light entering the camera is reflected from 3D objects visible from the camera. Therefore, rendering is concerned only with objects visible from the camera and the amount of light falling on those. Once 3D objects visible from the camera are found, their colors are computed. The color of an object depends on its properties and the amount of light falling on it. Therefore, the process of computing object color can be seen as computing incoming light on the object's surface and computing how is light reflected based on the object's properties. That said, rendering can be fundamentally divided into: (1) *visibility solving*, (2) *shading and light transport*.

Visibility Solving

Visibility-solving (Figure 2.34) is a process of determining which 3D objects, based on their geometry and transformation, are visible from the camera. More generally, if any two points in a 3D scene are visible to each other. This generalized view on visibility will later be useful when discussing light transport.

There are two main visibility-solving methods used in practice: ray-

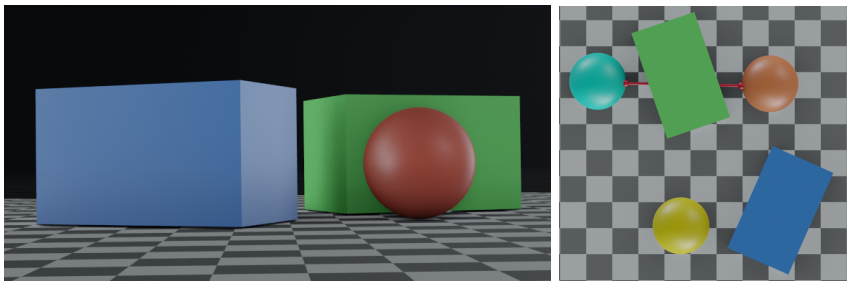


Figure 2.34: Left: objects visible from camera position. Right: corresponding 3D scene viewed from "top". Note how yellow and cyan spheres are not visible from the camera position. Further, the visibility can be generalized to any two points. Note how the red points connected by red line on the cyan and orange spheres are not visible to each other. This implies that the red point on the orange sphere can not reflect light on the red point on the cyan sphere and vice versa.

tracing and rasterization. *Ray-tracing* approach is directly inspired by geometric optics of light flow and thus more suitable for synthesizing complex light phenomena. On the other hand, *rasterization* utilizes the concept of mapping 3D geometry onto a 2D image plane making it extremely efficient.

The ray-tracing approach (Figure 2.37) first generates *camera rays* for each pixel of the image using camera information. Next, intersections of generated camera rays with 3D objects in a scene are found. Finally, for each intersection point, shading is computed resulting in color which represents the color of the pixel from which the ray was generated. Therefore, ray-tracing is called image-centric.

Rasterization is taking the reversed approach and thus is known as the object-centric approach. Rasterization assumes that the geometry of all 3D objects is triangulated mesh. Each triangle is projected and all pixels are traversed and tested if are inside the triangle. In the simplest case, the pixel is considered inside of a triangle if the pixel center is inside a triangle. Finally, if the pixel is inside a triangle, shading is computed resulting in color of that pixel.

Both ray-tracing and rasterization have advantages and disadvantages, which will be discussed in Sections 2.6.1 and 2.6.2, thus their choice depends on application requirements.

Shading and Light Transport

Shading is the process of computing the color of 3D objects visible from the camera. Therefore, shading is invoked after visibility-solving computation. Described visibility solving using ray-tracing or rasterization can be seen as sampling the surface resulting in surface position p. Thus, shading is

calculated for each surface point p. The color of surface point p is described with the rendering equation discussed by Kajiya [75].

The rendering equation is very general covering different light-surface interaction scenarios. Limiting ourselves on opaque, reflective surfaces, color of a surface in a point p is described with reflectance equation (Equation 2.9) discussed by Pharr [14], ch. 5.

$$L_o(p, \omega_o) = \int_\Omega f(p, \omega_o, \omega_i) L_i(p, \omega_i)(\omega_i n) d\omega_i \tag{2.9}$$

Intuitively, reflectance equation describes the color $L_o(p, \omega_o)$ in surface point p viewed from direction ω_o. This color is integral of incoming light from directions ω_i over the hemisphere Ω above the surface, placed around surface normal n, modulated by surface properties. Each incoming light $L_i(p, \omega_i)$ on surface point p from direction ω_i is attenuated by surface orientation n towards light ω_i and multiplied with BRDF $f(p, \omega_o, \omega_i)$ which determines amount of light coming from direction ω_i into direction ω_o.

Main components of the reflectance equation are incoming light $L_i(p, \omega_i)$ on surface point p, amount of light reflected from surface point p described with BRDF $f(p, \omega_o, \omega_i)$ and attenuation factor $\omega_i n$ describing attenuation of light due to surface orientation, that is its normal n, towards incoming light ω_i. For each surface point p, the BRDF parameters are set using texture for evaluation. Therefore both BRDF and texture form the material description of a 3D object.

To compute the amount of outgoing light $L_o(p, \omega_o)$ reflected from the object surface point p into the view direction ω_o, it is required to compute incoming light $L_i(p, \omega_i)$ to surface point p. Incoming light $L_i(p, \omega_i)$ can be computed using different *light transport* methods. Generally, incoming light (illumination) can be divided into direct or indirect (Figure 2.35). *Direct illumination* describes light coming directly from a light source to the surface point p. *Indirect illumination* describes light coming from any direction, e.g., reflected from another surface to surface point p. Direct and indirect illumination combined result in *global illumination*.

Solving indirect illumination represents a hard problem since it requires computing light from any direction to the surface point. The light can be, for example, multiple times reflected from other surfaces until reaching the surface point p. For each surface contributing, light can also come from any direction, thus implying the recursive nature of the rendering equation. To systematically approach this problem Heckbert [76] introduced light paths notation. Heckbert's notation distinguishes between different types of surface reflections along the path. The possible vertices of the light path are the light source (L), camera (E), diffuse reflection (D) and specular reflection (S). Vertices such as D and S can appear zero, one or multiple times.

Two main approaches for solving indirect illumination and thus global illumination are Monte-Carlo methods based on ray-tracing or finite elements

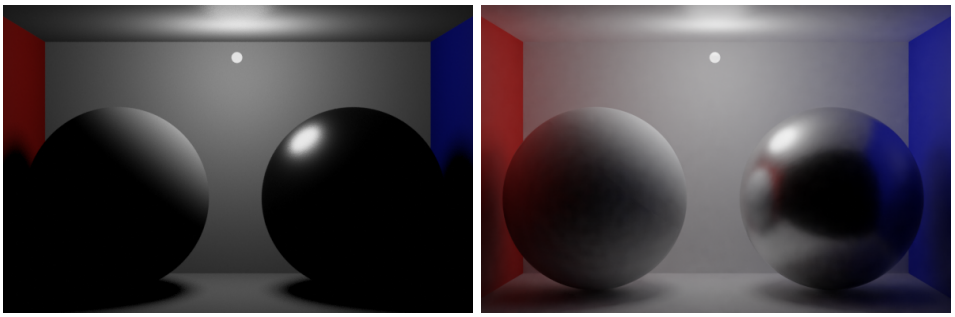


Figure 2.35: Diffuse and glossy spheres rendered using: only direct illumination (left), direct and indirect (global) illumination (right). Note how in the case of direct illumination only light source illumination is used to shade the surface. Further, note how all surfaces which are not visible to the light source are completely black. Note how in the case of indirect illumination, other surfaces contribute to shading visible as reflection.

methods. State-of-the-art Monte-Carlo ray-tracing-based method for light transport is called path-tracing discussed by Kajiya [75]. Path-tracing is an elegant method which can solve light transport for diffuse (D) and specular (S) surfaces. As such, it represents a unified light transport method capable of completely describing light transport in a 3D scene. On the other hand, state of the art finite elements method is called radiosity discussed by Goral et al. [77], which accounts only for diffuse surfaces.

For the sake of completeness, it is worth mentioning that light transport methods, e.g., path tracing can be extended to volumetric light transport for simulating light interaction for SSS and participating media as discussed by Fong et al. [32]. Since the focus in on surface reflection, volumetric 3D objects are out of the scope of this thesis.

Practical Considerations

When it comes to rendering in practice, ray-tracing or rasterization are often used for visibility solving. When it comes to shading, every rendering procedure aims to solve the rendering equation to a certain extent introducing various simplifications. Often, those simplifications aim to make the computation of incoming light $L_i(p, \omega_i)$ (Eq. 2.9) more tractable and efficient.

As mentioned, rendering can be seen as a process of discretizing a 3D scene into a 2D image. A 2D image is a two-dimensional array of pixels (discrete picture elements) with discrete intensity levels. That said, rendering must compute the color of each pixel in a 2D image. Each pixel of a 2D image covers a certain portion of a 3D scene called pixel footprint. Pixel footprint often contains complex information and yet pixels can represent only one color. This clearly shows the problem of discretization. For de-

tailed 3D scenes, discretization often leads to aliasing. *Aliasing* is visible as artefacts on a rendered image, that is, elements that do not exist in a 3D scene due to poor sampling and reconstruction as discussed by Pharr et al. [14], ch. 7.

Aliasing artefacts are reduced using various *anti-aliasing* methods. In the case of overly detailed textures which are used on 3D object surfaces, the anti-aliasing can be reduced using *mip-mapping* which aims to obtain a one-to-one pixel-to-texel ratio as discussed by Ewins et al. [61]. The source of aliasing can be any 3D scene element which contains high frequency or sharp transitions. Therefore, the most efficient anti-aliasing method is to use multiple samples per pixel (e.g., multiple rays per pixel) to compute a color which approximates the pixel footprint the best as discussed by Pharr et al. [14], ch. 7.

2.6.1 Ray-tracing-based Rendering

Ray-tracing is a method for solving visibility - determining if two points in a 3D scene are visible to each other. Rendering based on ray-tracing is inspired by physical light propagation as described by geometric optics. Therefore, light is represented as a ray. Ray-tracing is used to find the geometry of 3D objects visible from the camera. Further, it is used for light transport, for computing incoming light on the object surface.

Since ray-tracing-based rendering is simulating real-world light flow it is capable of producing photo-realistic images (Figure 2.36). That said, it can produce various phenomena such as soft shadows, occlusion shadows, reflections, transmissions, etc. For this reason, the image synthesis workflow in this thesis is based on ray-tracing, that is, a path-tracing rendering procedure.

The ray-tracing-based rendering can be explained using two functions: *trace*() and *shade*() shown in Figure 2.37 and Algorithm 1 (Akenine et al. [1], ch. 26). Function *trace*() is responsible for finding the closest intersection point given a ray and the objects in a 3D scene. Function *shade*() is responsible for computing the color of the intersection point found by *trace*().

Rendering starts by generating rays from the camera to each pixel of the image. These rays are called *camera or primary rays*. Function *trace*() tests camera ray intersection with objects in a 3D scene and returns the closest intersection. Function *shade*() is then called to calculate the color in the found intersection point. Since color depends on incoming light, function *trace*() is used for computing the incoming light on the intersection point. In the case of direct illumination, function *trace*() can be used for testing if the intersected point is visible to the light source. In the case of indirect illumination, recursive invoking of *trace*() and *shade*() functions is performed to compute incoming light reflected from other surfaces. After

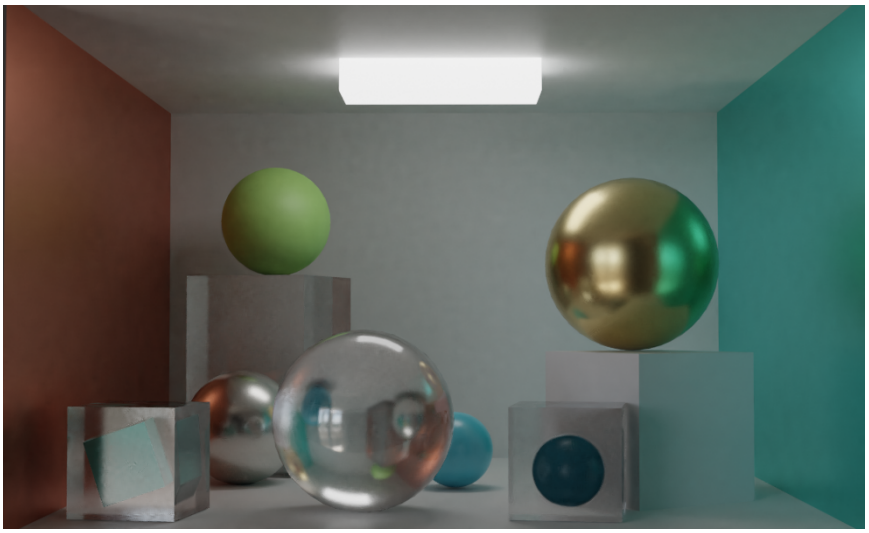


Figure 2.36: 3D scene lit using physical and environment light, containing various complex surface materials: plastic, glass, glossy metal. Note how ray-tracing-based rendering can produce realistic images containing soft shadows, transmission, inter-reflections, indirect illumination, etc. which would be hard to render using rasterization.

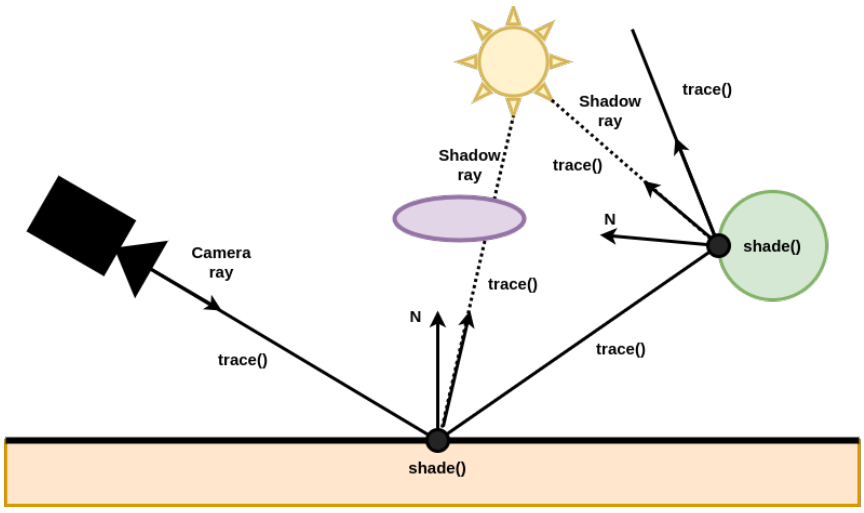


Figure 2.37: For each pixel, a camera ray is created. Function *trace*() is called for ray-tracing using the camera ray. Function *shade*() is called for the found intersection point to compute the color. To do so, function *shade*() calls function *trace*() while evaluating BRDF at this point. Therefore, function *trace*() is called to ray-trace shadow ray and reflection ray. This process is recursively continued. The figure is recreated based on Akenine et al. [1], ch. 26

incoming light is computed, function *shade*() uses material information: BRDF and texture to compute the color of the camera ray intersection. Computed color is then used as pixel color for which the camera ray was generated.

Algorithm 1 Ray-tracing based rendering

```
function rayTraceImage( )
    for p in image pixels do
        color of p = trace(camera ray through p);
function trace(ray)
    pt = find closest intersection;
    return shade (pt);
function shade(point)
    color = 0;
    for L in light sources do
        trace(shadow ray to L);
        color += evaluate BRDF
    color += trace(reflection ray);
    color += trace(refraction ray);
    return color;
```

Based on this discussion, ray-tracing-based rendering can be described with the following steps:

- Generating camera rays which will be used for visibility solving.
- Testing the intersection of camera rays with the geometry of 3D objects to find the closest intersection. This step effectively finds 3D objects visible from a camera.
- Computing color, i.e., shading, of the closest intersection point by using object shape and material information as well as incoming light computed with light transport.

Generating Camera Rays

Before discussing the generation of camera rays, it is required to define a *ray*. Ray is defined with origin O - a three-dimensional point and direction D - a three-dimensional vector. Any point on a ray is defined with a parametric equation as shown in Equation 2.10.

$$P(t) = O + t * D \tag{2.10}$$

The camera ray is generated by using the camera aperture as the ray origin. Camera ray direction is calculated by connecting the camera aperture with the centre of each pixel on the camera's film (i.e., image) plane.

Often, generating one camera ray per pixel is not satisfactory due to aliasing. Therefore, pixel area is sampled and multiple rays are generated for each pixel sample. Finally, the computed camera ray must be transformed into the world space where all other objects in the 3D scene reside.

Testing Camera-ray for Intersections

Once the camera ray is generated, it is tested for intersection with the geometry of 3D objects. Two important implications have to be considered. First, geometry (i.e., shape) representation for efficient ray intersection testing. Second, acceleration of intersection testing for complex 3D scenes containing large amounts of complex geometries.

A wide range of different geometry representations exist as discussed in Section 2.2. The most widely used geometry representation for efficient rendering is triangle mesh. Triangle mesh can be seen as an array of triangles which can be independently tested for intersection, not caring about the actual shape. The ray-triangle intersection can be computed very efficiently using the edge function as discussed by [14], ch. 3. Finally, the advantage of triangle meshes is that a wide range of shape representations can be triangulated efficiently.

Even when efficient ray-geometry testing is achieved, production scenes and complex objects often contain a large number of triangles, making the overall rendering process slow due to a large number of intersection tests. To accelerate intersection testing, acceleration spatial data structures are used as discussed by Akenine [1], ch. 19 and Pharr [14], ch. 4. Acceleration data structures lower the amount of ray-triangle intersection tests by discarding regions of 3D scenes which are not relevant to the current ray. Acceleration data structures are often hierarchical and are organizing 3D scenes into tree-like structures. Root defines the whole scene while children define their volume which further contains children. Instead of testing intersections with all triangles in the 3D scene, the spatial data structure is traversed and only triangles inside the volume of interest are tested for intersection. For n objects, compared to naive intersection testing, such data structures give improvement from $O(n)$ to $O(log(n))$.

Acceleration data structures can be roughly separated into spatial subdivisions and object subdivisions. *Spatial subdivision algorithms* subdivide 3D space into regions and record which triangles are falling in which region. During intersection testing, only triangles inside the region through ray passes are tested. *Object subdivision* is progressively decomposing objects in a 3D scene into smaller sets of objects. During intersection testing, only triangles inside the bounding volume which is intersected are tested for intersection. Both spatial and object subdivisions successfully accelerate the general problem of ray intersection testing. Often used spatial subdivision algorithms are uniform grids and axis-aligned binary space partitioning

(BSP) trees. On the other hand, the often used object subdivision algorithm is bounding volume hierarchy (BVH).

Fujimoto [78] introduced *uniform grids*, a spatial subdivision approach where the 3D scene is decomposed into equally sized grid cells. Uniform grids are simple to implement and efficient to test for intersections. This approach is further extended with more efficient construction and intersection testing methods.

BSP trees are encoding subdivisions of the entire space which is adaptively subdivided with planes as discussed by Pharr et al. [14]. A BSP tree starts with a bounding box that encompasses the entire scene. If the number of primitives in the box is greater than some threshold, the box is split in half by a plane. Primitives are then associated with whichever half they overlap The splitting process continues recursively either until each leaf region in the resulting tree contains a sufficiently small number of primitives or until a maximum depth is reached. Splitting planes can be placed at arbitrary positions. The two most common variations of BSP trees are kd-trees [79] and octrees [80]. A kd-tree restricts the splitting plane to be perpendicular to one of the coordinate axes. The octree uses three axis-perpendicular planes to simultaneously split the box into eight regions at each step.

BVH enclose regions of space surrounding the objects with bounding volumes which are simple to test intersection against. Early BVH were based on grouping objects under a hierarchy of bounding volumes as discussed by Clark [81]. Each bounding volume surrounding the whole object is simpler to test for intersection compared to testing intersection on the object itself. Bounding volume is a simple shape (e.g., sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), etc.) which contains the tightest possible volume surrounding the object. During intersection testing, first, the bounding volume of an object is tested for intersection. If the ray intersects the volume then all triangles inside the volume are tested for intersection. Otherwise, all triangles of an object inside the bounding volume are skipped. Choosing the shape of the bounding volume is a trade-off between intersection test complexity and the number of intersections. Simpler shapes are easier to test for intersections but often do not fit the objects' shapes. This can lead to intersection testing of all triangles inside the bounding volume although the ray misses all triangles inside. On the other hand, complex bounding volumes fit the object more closely but testing them for intersection is more computationally expensive. More advanced BVH approaches further break object bounding volume into smaller volumes containing groups of triangles enabling more efficient intersection testing as discussed by Wald [82].

Shading and Light Transport

Once the closest intersection of the camera ray and object surface is found, shading is performed to compute the color in the intersection point. Therefore, a shaded intersection point is called a *shading point*. The color of the shading point can also be seen as light reflected from the object's surface into the camera. Therefore, color in the shading point is, as discussed, generally described with reflectance equation (Equation 2.9). That said, shading depends on the surface properties in the shading point (e.g., surface normal and material) as well as incoming light and viewing direction. Computed color in the shading point is assigned to the camera ray that was used for intersection testing, that is, to the pixel from which the ray was generated.

Given the incoming and outgoing direction in the shading point, the BRDF term in the reflectance equation gives information on how much light is reflected in the outgoing direction. In the case of shading camera-ray intersection, the outgoing direction is defined with the camera ray and is also called view direction. The direction of the incoming light is computed by tracing rays from the camera ray intersection point towards the light source. Using the same BRDF with constant parameters for each intersection point results in a smooth and overly perfect surface. Therefore, material description is only complete when texture over the surface is used to feed the BRDF parameters introducing spatial variation.

The attenuation factor in the reflectance equation depends on surface orientation, that is, the angle between the normal vector and incoming light direction. As discussed, in ray-tracing, the direction of the incoming light is computed by tracing rays from the camera ray intersection point towards the light source.

The crucial information that has to be computed is the amount and direction of incoming light. This information is computed using light transport. Incoming light can be separated into direct illumination and indirect illumination.

To compute *direct illumination*, a ray - shadow ray - is traced from the intersection point to all light sources in the 3D scene as discussed by Pharr [14], ch 14. The main task is to compute if there are light sources visible from the intersection - that is if no objects are obstructing the shadow ray. If the light source is visible, the amount of light and its direction is computed. This information can be elegantly computed since the light direction is determined with the shadow ray and the intersection of the shadow ray with the light source gives access to the light source emission information.

As discussed in Section 2.4 lights can be separated into physical and non-physical: point and directional lights. In the case of physical lights, as discussed by Pharr et al [14], ch. 14, the light shape area must be sampled. For each sample, a shadow ray is constructed connecting it to the

intersection point. If not obstructed, each of the shadow rays contributes to the intersection point light. In the case of point lights, a single shadow ray connecting the intersection point and point light is created and tested for intersections. In the case of directional lights, shadow ray can be generated in the direction of light and tested for intersections. If there are no intersections along the shadow ray, then the intersection point is visible to the light source. Therefore, shadow ray direction and light source color and intensity are used as incoming light information. If there are intersections along the shadow ray, for any type of light source, then the current intersection point is not receiving light (e.g., intersection with opaque object occurred) or receiving less light (e.g., intersection with partially transparent object occurred) for the current shadow ray. In this case, the intersection point is in shadow. This shows how direct illumination and shadows can be elegantly solved in ray-tracing-based rendering (Figure 2.24).

Besides *direct* illumination coming from light sources, light can come *indirectly* from any direction to a surface being shaded, e.g., after multiple bounces from other surfaces in a 3D scene. If only direct illumination is computed, surfaces which are not directly exposed to light are completely black as discussed by Pharr et al. [14]. Further, since specular and glossy surfaces would normally reflect their environment, without indirect illumination they would be mostly black except for the smaller area which reflects light source. Similarly, transparent surfaces refract and transmit light from their environment and thus would look unrealistic if only light from a light source would be refracted. Inter-reflections between surfaces can be separately tacked as *indirect specular* and *indirect diffuse*. Finally, computing indirect illumination enables the generation of soft shadows which would be completely black if not visible to light sources. That said, indirect illumination is a crucial component for realistic image synthesis (Figure 2.36).

An early light transport method based on ray-tracing is Whitted ray-tracing [83]. Whitted ray-tracing introduced an important concept of computing visibility *between surfaces* in 3D scenes which is a crucial element of light transport. Whitted ray-tracing computes visibility only between surfaces which are either ideal specular reflection (e.g., mirror-like) or ideal specular transmission. Therefore, after finding the camera ray intersection, rays are further traced bouncing between specular and transmissive surfaces until reaching the light source or diffuse surface. If the light source is reached then its color is simply used for computing the color of the shading point. If a diffuse surface is reached then its color computed by direct illumination is used for computing the color of the shading point.

Work done by Kajiya [75] can be seen as extending Whitted ray-tracing. Kajiya introduced the light transport equation (LTE) describing the equilibrium distribution of light in a scene. Further, in this work, Kajiya introduced path tracing, which is a general-purpose, unbiased Monte-Carlo light transport algorithm. Path-tracing simulates incoming light on camera-ray

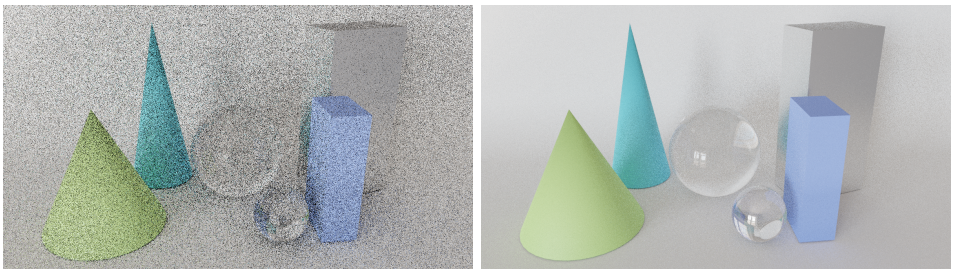


Figure 2.38: Images rendered using path tracing. Left: 1 sample per pixel. Right: 32 samples per pixel.

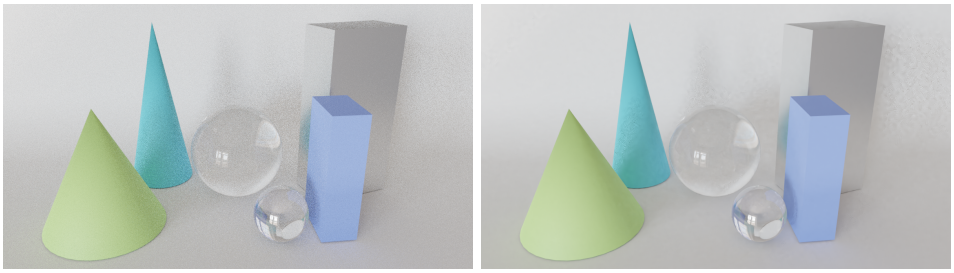


Figure 2.39: Images rendered using path tracing. Left: 32 samples per pixel. Right: 32 samples per pixel with denoising. Denoising removes noise, but can blur parts of the image depending on the noise level.

intersection point by computing complex light path through 3D scene to the light sources. This light path simulates light scattering across surfaces in 3D scene. Path-tracing uses Monte-Carlo sampling which can be seen as collecting all light which illuminates camera-ray intersection for all directions contained in the hemisphere above this intersection point. It is *unbiased* method meaning its only source of error comes from statistical variance which can be observed as noise in the rendered image. The process always approaches the most accurate result possible. It is general-purpose meaning that it can be applied for any type of surfaces and materials: specular, glossy, diffuse, transparent, translucent, participating media, etc. This makes path tracing one of the most elegant methods enabling practical realistic image synthesis based on ray-tracing and Monte-Carlo sampling. The main problem with path-tracing is that it requires a large number of samples to produce images with a low amount of noise (Figure 2.38). This implies significant computation time. An even larger number of samples and computation time is needed for complex scene setups such as a combination of transparent and diffuse surfaces causing caustics. Eliminating noise in complex scenarios, besides increasing samples and computational time, can be done using denoising (Figure 2.39) as discussed by Zwicker et al. [84].

2.6.2 Rasterization-based Rendering

Rasterization-based rendering, compared to ray-tracing-based rendering, is extremely fast and often used for real-time rendering. On the other hand, visual realism is sacrificed, thus complex light phenomena such as soft shadows, refraction, indirect diffuse and indirect specular can not be elegantly simulated as in ray-tracing-based rendering. Therefore, in this work, ray-tracing-based rendering is used for generating photo-realistic images, while rasterization-based rendering is used for an interactive virtual surface inspection planning environment discussed by Gospodnetic [3].

Rasterization, next to ray-tracing, is another technique for solving visibility. Although visibility solving via ray-tracing can be generalized for any two points in a 3D scene, visibility solving with rasterization is performed only from the camera's point of view. Therefore, only the first visible surfaces can be obtained efficiently and obtaining visibility between arbitrary two points is highly inefficient. However, obtaining the first visible surfaces using rasterization is extremely efficient as well as its efficiency for hardware-accelerated computation resulting in real-time performance. When it comes to shading of first visible surfaces, rasterization can not offer efficient visibility information and thus various complex light phenomena can not be simulated as efficiently as in ray-tracing-based rendering (Figure 2.40). For example, inter-reflections and soft shadows can not be efficiently implemented since a large amount of 3D scene information is discarded. Further, physical lights can not be simulated easily (Figure 2.24) while sampling their surface is not feasible due to a lack of 3D scene information. To conclude, rasterization can not be efficiently used for advanced light transport algorithms required for shading which reflects in amount of visual realism.

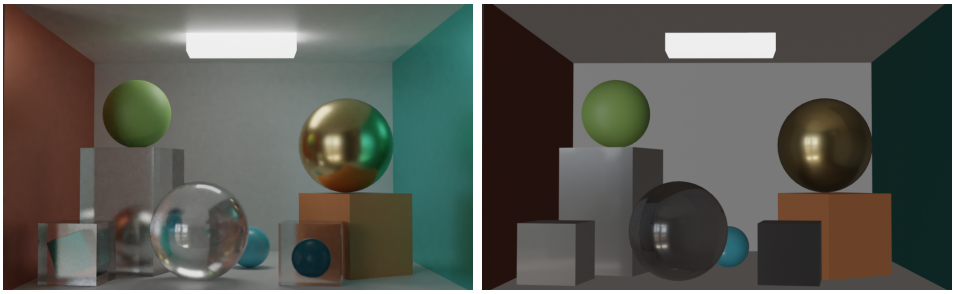


Figure 2.40: Complex scene lit with physical light and environment light. Left: ray-tracing-based rendering. Right: rasterization-based rendering.

Rasterization: Visibility Solving Method

Rasterization is a highly efficient method for solving visibility for 3D objects for which geometry representation is described with triangle meshes.

The rasterization process can be separated into two main stages: triangle projection and triangle rasterization as discussed by Akenine et al. [1], ch. 23.

The *triangle projection* step projects vertices of each triangle of each mesh object in a 3D scene onto an image plane. Triangle projection is preceded by multiple transformations which convert vertex coordinates from world space, in which all 3D objects reside, into a space suitable for projection on the image plane. Projection is then the final transformation described with the perspective projection matrix. After the projection step, each triangle vertex has two-dimensional coordinates called raster coordinates. Further, each vertex contains information about distance from the camera which is called depth or z-value.

In the *triangle rasterization* step, each pixel of the image plane is tested if it is contained inside a projected triangle. In the simplest case, the centre of a pixel can be used to determine if the pixel is inside a projected triangle. Determining if the pixel centre point is inside the triangle is called the *inside-outside test* which relies on the edge function as discussed by Akenine et al. [1], ch. 23. When rasterization is used during rendering, for pixels which are inside the triangle the color is calculated in the shading step. Similarly as discussed for ray-tracing, each pixel when projected into a 3D scene covers a certain area - *pixel footprint*. Pixel footprint can contain complex 3D scene information containing high-frequency details. To evade aliasing, multiple samples per pixel are often taken to compute a more accurate color estimation of pixel footprint.

Often, pixel samples can overlap multiple triangles. To resolve this, for each pixel sample, the distance from the camera to the corresponding triangle point is computed. Only points on the closest triangles are processed further, while others are discarded. For each pixel, this distance information is stored in a *depth buffer* - a 2D array which has the same size as the image. Once all triangles and all pixels are traversed, the depth buffer contains distances to the closest triangles from the camera. Information stored in a depth buffer is extensively used during rendering, especially during shading.

Rasterization-based Rendering

Rasterization-based rendering can be conceptually described using *graphics rendering pipeline* (Figure 2.41) which is commonly implemented on GPU using shader cores for hardware-accelerated rendering as discussed by Akenine et al. [1], ch. 2.

Two main tasks of the graphics rendering pipeline are visibility solving and shading. The *visibility solving* step is implementing rasterization to determine which objects are visible from the camera. The *shading* step is responsible for computing the color of visible objects. These tasks are solved in four conceptual stages: application stage, geometry processing stage, ras-

Figure 2.41: Graphics rendering pipeline.

terization stage and pixel processing stage as discussed by Akenine et al. [1], ch. 2. Optionally, compute shader can be used for GPU computation which is not necessary rendering.

Application stage. In the application stage, 3D scene information is defined: 3D objects, lights and cameras. Each 3D object is assigned with certain transformations defining its position in the 3D scene. Transformation on a 3D object is applied in the geometry processing stage. 3D object shapes are often described as triangle meshes. If a different geometry representation is used for describing the shape of the 3D object, then it is triangulated to obtain the triangle mesh which is further processed in subsequent stages. As discussed, the triangulated mesh is used for visibility solving in the rasterization stage. Shading information of 3D objects is defined as a shader program which is commonly invoked in the pixel processing stage. The shader program contains an algorithm known as *shading model* which completely describes the computation of the color of the 3D object. The shading model combines BRDF with texture information to describe the material and thus object's appearance. The shading model often implements direct illumination computation and thus uses information on light and camera in the 3D scene. Parameters of the shading model are stored as vertex attributes which are in subsequent stages interpolated across the triangle and used for shading. In rasterization-based rendering, often non-physical lights are used, such as point and directional lights, which can be efficiently incorporated into the shading model. For point lights, their position, intensity and color can be defined in the application stage and used in the shading model. For point lights, their direction, intensity and color can be defined in the application stage and used in the shading model. The camera defines a portion of visible 3D scene which are dependent on camera properties, location and orientation. In rasterization-based rendering, camera information is defined with transformation and projection matrices which are used during the rasterization stage. On the application stage, the interaction and animation of 3D objects is also defined. Complex tasks such as particle system behaviour, tessellation and simulations can be delegated to *compute shader* for faster computation on GPU. Information on the 3D scene which is defined on the application stage is sent to the next stage: geometry processing.

Geometry processing. The main purpose of the geometry processing stage is to prepare 3D objects for rasterization as discussed by Akenine et al. [1], ch.2. More specifically vertices of all triangle meshes of all 3D ob-

jects are transformed to be rasterized in the next stage. This stage is under the user's control meaning that additional transformations on vertices can be computed before triangle vertices are transformed and projected on the image plane for rasterization. For example, the user can specify different scaling, rotation and translation transformations applied to the same geometry which effectively copies or *instances* 3D objects across the 3D scene. Triangle vertices, in this stage, are transformed through local, world, camera and finally screen space. These transformations are specified with 3D object transformation, camera transformation and projection transformation (e.g., perspective projection). Geometry which is outside of camera frustum or which is occluded by other geometry, is clipped and culled enabling a more efficient rasterization step as well as the whole rendering process. Besides geometry computation, this stage can also be used for evaluating and setting up vertex attributes such as colors, normals or texture coordinates. For example, for each vertex, a shading model can be evaluated resulting in color. This color can be interpolated over triangles in subsequent stages. Shading in this stage is restricted only to vertices and thus more advanced surface effects such as variation of details can not be achieved. Optional geometry processing can be performed using tessellation shader, geometry shader and stream output but which are currently out of scope.

Rasterization stage, as discussed, is solving the visibility problem, i.e., which triangles are visible from the camera's point of view. More specifically, the rasterization stage is responsible for determining which pixels of the image plane are overlapping which triangles. Since triangle vertices are already projected on the image plane, this is done by looping over each triangle and over all image pixels. For each pixel, samples which are determined to be contained in the triangle are sent to pixel processing. The rasterization stage can be decomposed into triangle setup and triangle traversal. In the triangle setup step, all data needed for subsequent stages, such as edge function and barycentric coordinates, is computed. After which in the triangle traversal stage, each pixel is checked if covered by a triangle using the inside-outside test. For each part of the pixel which is covered by a triangle a *fragment* is generated. All fragments inside a triangle are sent further to the pixel processing stage.

Pixel processing can be decomposed into pixel shading and pixel merging sub-stages. The pixel shading stage computes the color of pixel samples which were found to be overlapping a triangle. Color is computed using a shading model which is defined as a shader program defined on the application stage. Evaluation of the shading model is done with interpolated vertex attributes for each point on the triangle corresponding to the current pixel. The result of the pixel shading stage is computed color for pixels samples, that is fragments. The pixel merging stage combines computed color per fragment into a pixel color. The pixel merging stage further enables various operations such as blending required for transparent surfaces, stencil

for controlling what is rendered and visibility resolving. Once all pixels are processed, the result is the image - *color buffer* which is a rectangular array of colors, ready for display.

Rasterization-based Rendering Optimizations

The disadvantage of the described rasterization-based rendering is that only information on the first visible surfaces from the camera is recorded while the rest of the 3D scene information is discarded which is also discussed in Christensen et al. [29]. This is because rasterization, which is used for visibility solving, is extremely efficient only for obtaining the first visible surfaces from the camera's point of view. This is also due to culling and clipping of occluded surfaces or surfaces out of the camera's frustum. Therefore, most of the 3D scene information is discarded and not available during shading which is performed in the pixel processing stage. Due to this limitation, computing indirect illumination requires additional data-structures storing the 3D scene information. On the other hand, the computation of direct illumination is much more tractable. Often non-physical lights (e.g., point lights) are used since the lack of information on surfaces in 3D scenes during shading, disables the possibility of computation with physical lights (Figure 2.24). Direct illumination is often computed by looping over all lights in a 3D scene. For each light, the shading model is evaluated and added as a contribution to the final color.

Lack of information on surfaces in 3D scenes significantly cripples possibilities of advanced light transport methods. Therefore, computing more advanced, indirect light-surface interaction for producing realistic effects such as shadows, soft shadows, inter-reflections and transparent surfaces is not possible directly (Figure 2.40) and requires additional techniques. Those techniques are rarely physically-based and always involve strong assumptions which often decline in particular complex 3D scene setups. However, they are inspired by complex light phenomena and can often be used to achieve the desired appearance. Often desired light phenomena in synthesized images are shadows (Figure 2.42). A wide range of methods are developed for shadow computation in rasterization-based rendering. Notable methods, discussed by Hasenfratz [85], are planar shadows, shadow volumes and shadow mapping. Further, there is a wide range of methods simulating additional indirect illumination effects. Notable methods are environment mapping [71] (Figure 2.43) and ambient occlusion (AO) [86] (Figure 2.44). Global illumination methods can be used to precompute the amount of light on surfaces in 3D scenes. This method is called *light baking* [87]. Precomputed light is stored in texture and used during rendering. Finally, simplifications of global illumination and the introduction of additional data structures enable dynamic computation of indirect light such as described in the voxel cone tracing approach discussed by Crassin et al. [88].

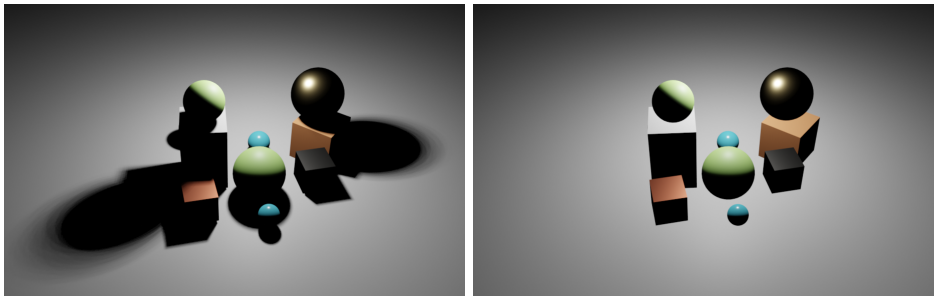


Figure 2.42: Rasterization-based rendering. Left: with shadow. Right: without shadow.

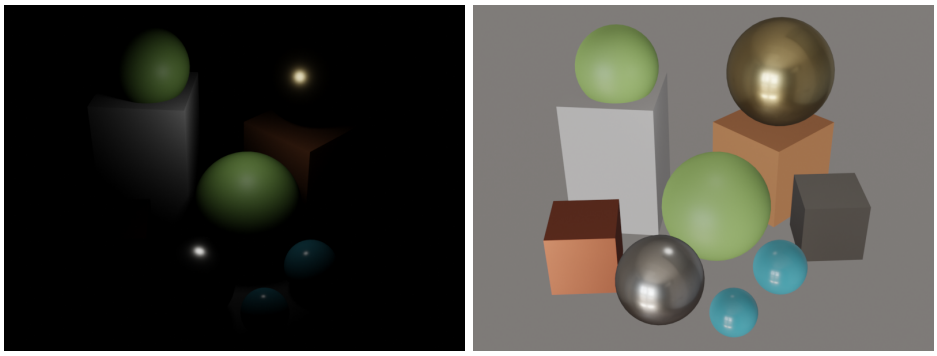


Figure 2.43: Rasterization-based rendering. Left: only non-physical point light. Right: environment illumination and non-physical point light.

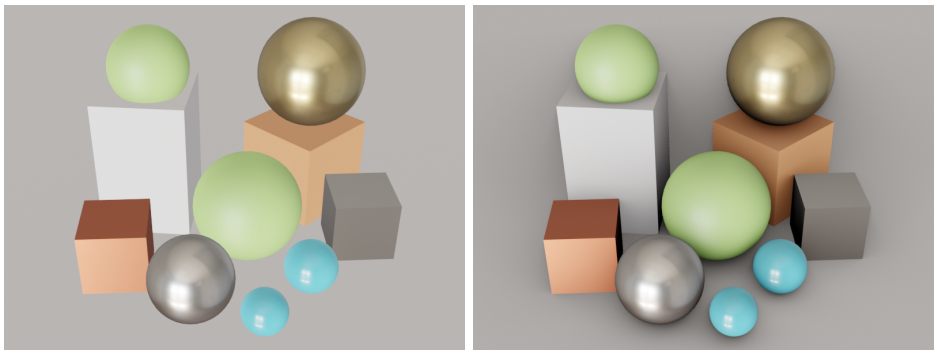


Figure 2.44: Rasterization-based rendering. Left: without ambient occlusion. Right: with ambient occlusion.

The described rasterization-based rendering pipeline performs so-called *forward shading*. Although intuitive and simple, forward shading is very expensive for a larger number of light sources. To solve this problem, a different paradigm called *deferred shading* was introduced as discussed by Akenine et al. [1], ch. 20. Deferred shading uses *multiple render targets* technique to store different scene information from the camera's point of view in *geometry buffer*, short *G-buffer*. Information stored in the G-buffer is then used for efficient computation of the final image. Since only information on the first visible surfaces is stored in the G-buffer, the particular disadvantage of deferred shading is the rendering of transparent surfaces. To solve the rendering of transparent surfaces, forward shading can be combined with deferred shading. In this scenario, forward shading is used to compute the color of transparent surfaces while opaque surfaces are shaded and using deferred shading.

2.7 Image and Display

After rendering, the image is often additionally processed before further usage or displaying on a display device. Operations that are performed on an image are described with *imaging pipeline*, discussed by Akenine et al. [1] et al, ch. 5. The imaging pipeline contains three main steps: user-defined post-processing, tone-mapping and display encoding.

Post-processing

Post-processing is either used to introduce effects which are too expensive to simulate during rendering or to achieve certain visual styles. Depending on the image synthesis environment environment, post-processing can be applied be applied on-the-fly [89] (e.g., real-time rendering) or on images of already rendered frames [90] (e.g., offline rendering).

Post-processing can be performed by applying image processing techniques such as blurring, edge detection, dilatation, quantization, pixelization, etc. [91] (Figure 2.45) on the rendered image (i.e., color buffer). Further, additional 3D scene information can be computed in multipass rendering and stored such as depth buffer, normal vectors buffer, velocity buffer, etc. These buffers provide additional insight into 3D scenes and enable advanced effects such as glare (i.e., bloom, lens flare), depth of field, motion blur, etc. [89] (Figure 2.45).

Tone-mapping

During rendering, a 3D scene is discretized in order to obtain a 2D image for visualization on display devices. Display devices, besides limited spatial resolution, are limited in color display capabilities. This means that colors

Figure 2.45: Left: rendered image. Middle: blur post-processing. Right: glare effects post processing.

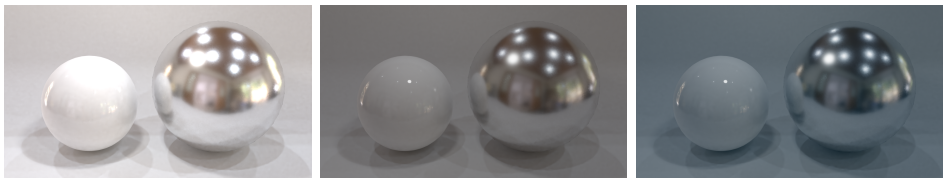


Figure 2.46: Left: rendered image. Middle: tone mapping (strong contrast reduction, approximating high dynamic range for limited dynamic range display). Right: color grading (artistic manipulation of image color).

of the rendered image will not look as expected once visualized on display devices. For this purpose tone mapping is performed as discussed by Durand et al. [92]. Tone mapping is concerned with *image reproduction* and *preferred image reproduction*. Image reproduction aims to create an image which gives observers the same impression as if they were observing the original scene. Tone mapping can be seen as process of converting rendered image colors to display device color values. Image reproduction can be performed in two steps: scaling the image by exposure and fitting the dynamic range of a rendered image to the dynamic range of the display device using tone reproduction transform such as Reinhard transform [93]. On the other hand, *preferred image reproduction*, that is, *color grading* is creative manipulation of image color to achieve the desired artistic look as discussed by Faridul et al. [94] (Figure 2.46).

Display Encoding

The final operation that has to be performed on the rendered image, before displaying it, is encoding its color values into display values. Until this step, image color values are in *linear color space*. Linear color space is needed for correct rendering computation and manipulation of image colors. However, the relationship between input color values and display device light intensity is described with *display transfer function* which is non-linear as discussed by Akenine et al. [1] et al, ch. 5. Therefore, linear color values are encoded for display by applying the inverse display transfer function. This operation is known as *gamma correction* [95], which applied on linear color space results in *sRGB* color space for which most computer monitors are

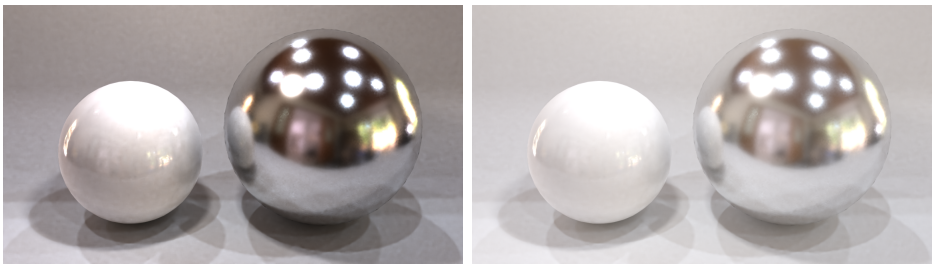


Figure 2.47: Left: linear color values resulting from rendering. Right: gamma correction, sRGB color values. Note how linear color space appears dimmer since the non-linear display transform is not cancelled out.

configured. Displaying gamma-corrected image color values cancels out the non-linearity of the display device transfer function (Figure 2.47).

Chapter 3

Procedural Modeling Background

Procedural modeling is a paradigm in which 3D scene elements such as geometries, textures, lights, shading, animation, etc. (Figure 3.1, 3.2) are algorithmically defined (i.e., using code, procedures). As such, procedural modeling perfectly fits for creating complex and thus realistic shapes as well as highly detailed patterns that would be hard and time-consuming to create manually. Procedural modeling is highly beneficial when it comes to modeling highly complex natural objects or phenomena: surface or interior of a stone, wood, marble, sea surface, clouds, smoke, fire, landscapes, etc. as discussed by Ebert et al. [62]. Further, this approach is valuable for modeling man-made elements which are complex but contain clear structure and order such as buildings, cities, roads, manufactured surfaces, etc. as discussed by Smelik et al. [96].

The term *procedural* is used to denote elements that are described by code rather than data structure (i.e., procedural vs declarative) as discussed by Ebert et al. [62]. However, almost every procedure takes some parameters as input and can rely on certain data and data-structures The defining characteristic of procedural elements is that they are synthetic - generated by a program rather than painting or digitizing the real-world. Thus, procedural modeling can include mathematical models, phenomenological approaches, physical simulation, artificial intelligence (e.g., machine learning), etc.

Procedural modeling results in parameterized elements of a 3D scene, e.g., parameterized texture or geometry model which is evaluated with fixed parameters to generate the specific instance of a 3D scene element (Figure 3.1, 3.2). This way of modeling enables controllable, automated and repeatable creation of 3D scene elements - from their diversity of types to their sheer number as discussed by Ebert et al. [62] and Raistrick et al. [97].

Often, procedural modeling is criticised because of the experience required, compared to manual or data-based modeling, for writing a procedure

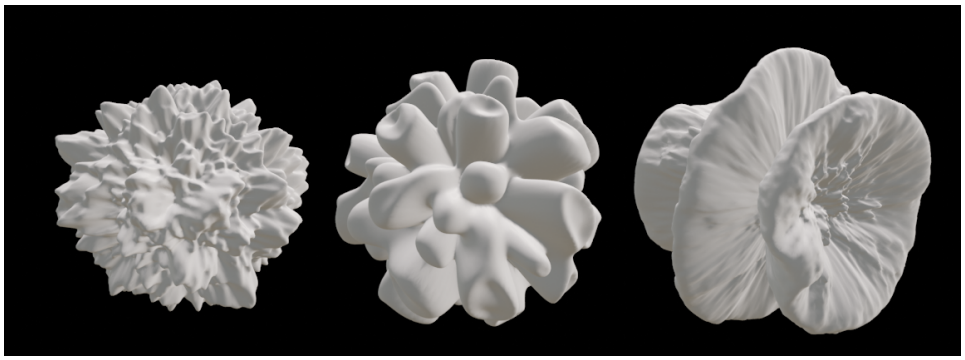


Figure 3.1: Procedurally generated geometrical shapes. By only changing the parameters, different instances of geometrical shapes can be generated.

as well as non-intuitive parameters for obtaining the desired look. On the other hand, procedural modeling enables easier creation of complex visual elements (e.g., rich surface details) which are often required for photo-realistic image synthesis as discussed by Ebert [62]. Procedural modeling can be coupled with other modeling methods. For example, physical simulation can be used to compute the behaviour of 3D scene objects where the procedural part computes the driving forces of the simulation. Another example is using artificial intelligence (AI) (e.g., generative deep learning) and data-based approaches (e.g., image data) for more tractable development of procedural models as discussed by Guerrero et al. [98]. Further, the development of procedural models is more tractable when computation is performed on input providing the information about the target domain. For example, if a procedural model for generating texture on a 3D object is developed, then 3D object geometry information such as surface tangent field can be used as input to the model making the development of texture patterns on arbitrary surfaces more tractable. Another example is the interactive usage of the procedural model where the user is guiding the computation by providing the spatial input, e.g., position on a 3D object. This way, a powerful creative process is possible as discussed in [99], [100], [101], [102], [103].

That said, procedural modeling represents a paradigm which holds irreplaceable characteristics and thus found its usage in a wide range of applications for modeling virtual environments as surveyed by Smelik et al. [96], Freiknecht et al. [104], Freiknecht [105], Emilien et al. [99], Ebert et al. [62], Kelly et al. [106]. The power of procedural modeling is adapted in 3D content creation such as films, games and virtual environments. A notable example of a tool for general-purpose procedural modeling for content creation, deeply integrating principles of proceduralism into the workflow is Houdini [107].

Although any 3D scene element can be procedurally defined, for the purpose of this thesis, the focus will specifically be on the procedural modeling

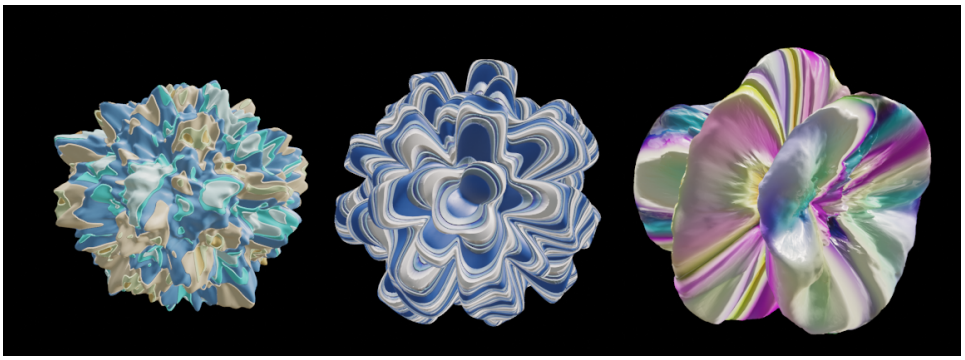


Figure 3.2: Procedurally generated texture on procedurally generated geometrical shapes. By only changing the parameters, a wide range of textures can be achieved.

of 3D object surface which includes surface texture (Section 3.1) and 3D geometrical surface defects (Section 3.2).

3.1 Procedural Texture Modeling

Realistic image synthesis heavily relies on surface shading - the process of computing color of the 3D object's surface (Chapter 2). Shading further relies on a surface reflection model (e.g., BRDF) and texture. The texture is used to vary the reflection model properties over the surface of a 3D object. Therefore, it is crucial for achieving realistic surface details and thus realistic appearance. In this chapter the focus is on procedural texturing - an approach where an algorithm is used to generate a texture [10].

Procedural texturing involves two steps: *authoring step* and *generation step*. During the authoring step, a parameterized procedural model is being developed (Section 3.1.1). During the generation step, the procedural model, developed in the first step, is evaluated with specific parameters to produce the specific instance of a texture (Section 3.1.2).

There are many advantages of the procedural texturing paradigm. First, procedural textures are extremely compact, reducing the need for storage of high-resolution texture maps. Since a procedural texture model must be evaluated to generate a texture, only code and parameters need to be stored. Second, procedural texture does not have a fixed resolution. Since a procedural texture model can be evaluated for any point in space no matter how large or small it is, the resulting texture will always appear detailed. This way, patterns described with procedural texture are of high quality regardless of observation distance. Furthermore, the procedural texture does not have a fixed area. The procedural texture model can be evaluated for any point in the space and thus can cover an arbitrarily large area with-

out seams or repetition artefacts. Finally, the procedural texture model is parameterized enabling the automated and controllable creation of a large number of texture instances.

However, there are several downsides to the procedural texturing paradigm. First, procedural texture models can be difficult to create. Developing a procedure which describes a complex texture pattern can be hard. This is especially prominent while developing the implicit procedural model as it will be discussed in Section 3.1.1. Second, procedural texture can be non-intuitive to control compared to scanning or painting a texture. Finally, the procedural texture model must be evaluated during rendering for each shading point which can be slower than simple raster image access.

3.1.1 Authoring Phase

Procedural texture modeling is often performed by artists or computer graphic experts who have a lot of experience, knowledge and intuitive reasoning as discussed by Ebert et al. [62]. Authoring a procedural model requires designing and developing a procedure which describes the desired texture pattern. Terms *procedural* and *texture* are very general therefore this process is still an art form: there is not a recipe that can be reused. Therefore, in this chapter, strategies, approaches, methods, techniques and building blocks which are combined, extended or built on are discussed (Sections 3.1.1, 3.1.1 and 3.1.1). By combining modeling strategies, and regular and irregular procedural modeling building blocks, a wide range of patterns ranging from natural to manufactured can be described. Manufactured surfaces have a certain degree of regularity due to precise manufacturing processes but also certain irregularities due to manufacturing errors and handling. Therefore, often the goal is to create a structured pattern with certain regularity and add irregularity to introduce imperfections and a more realistic appearance. To do so, artists and computer graphic experts rely on the fact that many textures are variants of each other, built by the same building blocks and concepts.

Describing a pattern can be done both using code or visual coding such as node-graphs (both are equivalent in terms of functionality). The texture pattern described in this way offers parameters which drive the procedural model. This way, the texturing model is controllable and enables obtaining the desired look and feel of the texture. It is important to note that input parameters can be scalars, vectors, arrays, images, etc.

Texture patterns can be extremely complex. Therefore, the first step is to determine the class of texture patterns to be modeled by a procedural texture model. Desired patterns in the class should be obtainable by tuning the parameters of the procedural texture model that is being created. Next, visual observation and analysis are performed to extract important texture features, texture elements and their composition. In this step, the pattern

is conceptually decomposed into a set of simpler shapes and their positions, directions, etc. Finally, the procedural texture model is developed by combining or extending various strategies, methods, approaches and techniques discussed in Sections 3.1.1, 3.1.1 and 3.1.1.

The development of a procedural texture model depends on several design decisions:

- Which kind of input to the procedural model can be expected? If certain information can be pre-computed and inputted to the procedural model, then certain operations do not need to be implemented in procedural model itself.

- Does the procedural model must be explicit or implicit (see Section 3.1.2)? Explicit models make writing easier since all geometrical and topological information can be accessed at any point. On the other hand, implicit models are invoked for any point in space randomly and geometrical or topological information is not (if not precomputed) available, except for the current shading point (e.g., position and normal).

- Is texture pattern anisotropic or isotropic? Certain patterns exist only in 2D space. For example, planar product surfaces can exhibit texture which is a result of a machining process that can be applied only on planar surfaces. Those patterns are often directed and anisotropic. On the other hand, certain patterns can be modeled in 3D space. Often, isotropic patterns can be well-defined in 3D space.

- Is texture pattern global or stationary, local? Global patterns vary spatially, while stationary patterns have more or less uniform structure over the area.

Discussed approaches to procedural texture model creation and design questions can further be specialized for specific applications and patterns aimed to be recreated. For the purpose of this thesis, the focus is on machined surfaces. Machining is a standardized and deterministic process producing standardized surface textures (i.e., finishing) as discussed by Black et al. [2]. Thus, classes of possible textures are bounded by machining processes. Once a class of texture pattern is selected, observation of textures and analysis of texture features can greatly benefit from knowledge of machining processes. For example, machined surface textures can be divided into isotropic (e.g., sandblasting, shot blasting, etc.) or anisotropic (e.g., grooves and directed lines which are caused by machining tool movement). Further, visual observation, image processing analysis and machine knowledge are used to extract information on texture pattern elements and their composition (e.g., lines or circles arranged or directed in a certain way). Finally, the extracted information gives clues on which strategy, methods,

approaches, and techniques to use for the development of the procedural texture model. That said, procedural texturing approach described in Chapter 5 can be conceptually decomposed in the following steps (Figure 3.3):

1. Texture classification: determine the type of machined surfaces to be modeled

2. Texture observation and analysis: use observation, image processing and surface machining knowledge to extract texture features

3. Texture model development: determine and use strategies, approaches, methods, and techniques for modeling the pattern

4. Texture model application: develop additional procedures needed to map procedural patterns on the 3D object

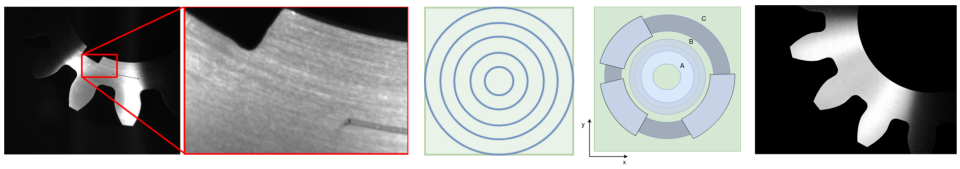


Figure 3.3: Circular brushing procedural texture workflow. First, a circular brushing pattern present on a gear object (left) is analysed. Next, the texture is conceptually decomposed into a set of circles which can be modeled using torus elements with varying widths (middle). Finally, with the decomposition of texture in mind, the procedural model is developed, applied on a 3D object surface and rendered (right).

Procedural Texture Modeling Strategies

Procedural modeling strategies in one or another way build upon a concept of simplification. The general strategy is to decompose the complex texture pattern into a set of simpler patterns which can be modeled more easily and combined to form a complex pattern.

Layering approach, stacks simpler patterns one on top of another as discussed by Ebert et al. [62], ch 2 and Perlin [11]. This approach can be seen as combining multiple layers of patterns to produce more complex patterns (Figure 3.4). Combining layers can be performed using parametrized and thus controllable functions. An example of layer combination is weighting using linear interpolation.

Composition approach takes several simpler functions and composes them to produce more complex functions. An example of function composition is taking a simple function which generates random numbers and

Figure 3.4: Two simple procedural textures patterns are present one left and middle sphere. The right sphere contains combined (layered) patterns.

using it as an input to another function which generates different colors for different inputs as discussed by [62], ch.2 and Perlin [11].

Masking approach uses one or more masks which define regions per object surface. Each mask can be procedurally defined, enabling control over the size, shape and positions of regions. This way, multiple textures can be layered on the surface while determining their influence using the mask assigned to each texture (Figure 3.5).

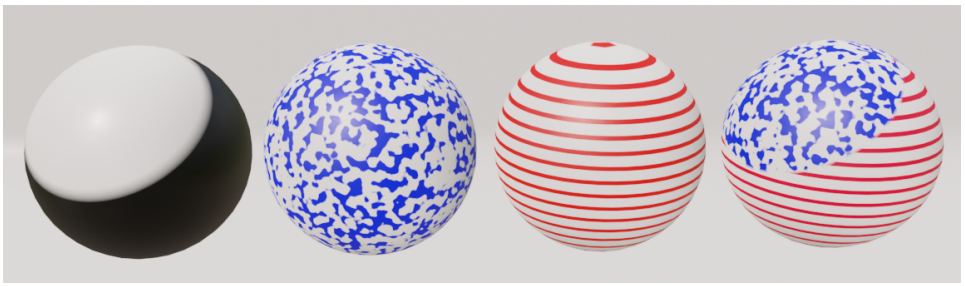


Figure 3.5: First sphere contains procedural texture which represents a mask. The second and third spheres contain two different procedural textures. The fourth sphere contains textures from the second and third spheres combined using the mask shown on the first sphere.

Texture elements and placement approach is inspired by *germ-grain model* [108], *random pattern placement* [62], ch. 2 and *texture bombing* [109]. The key idea is that texture patterns can be decomposed into texture elements and the placement of those elements (Figure 3.6). For example, a scratched surface can be seen as a collection of scratches (texture elements), which are placed in a certain way on the surface (placement of texture elements). This decomposition is useful since the shape of texture elements and their placement can be modeled separately. This approach is intuitive and easy for explicit procedural texturing models (Section 3.1.2). On the

other hand, placement modeling for implicit procedural texturing models (Section 3.1.2) is more difficult. This is because spatial information in the implicit procedural model is not inherently known. One solution to generate positions for placement is precomputing points and storing them as an image or a table. Another solution is to use implicit noise functions for generating the positions.

Figure 3.6: Placement of different texture elements over the object surface..

Domain warping approach takes a simpler pattern and perturbs the texture space, which in turn distorts (i.e., warps, deforms) the starting simpler, pattern into a new, more complex pattern (Figure 3.7) as discussed by Ebert et al. [62], ch. 2 and Quilez [110]. If the texture is defined as a function of space as $f(p)$, then warping is applying certain function $g(p)$ before evaluating the texture $f(g(p))$. For example, function $g(p) = p+h(p)$, where $h(p)$ is a small arbitrary distortion. This process can be seen as applying deformations on a pattern such as pinching, stretching, twisting, bending, re-scaling, etc.

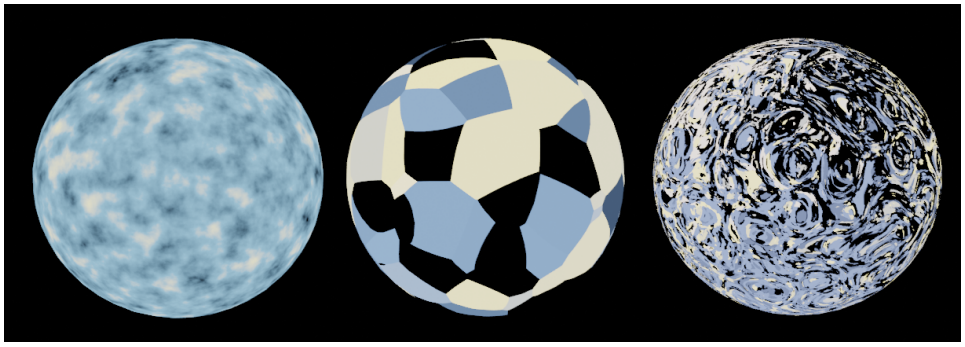


Figure 3.7: Left and middle sphere show two procedural textures evaluated using sphere surface points. The right sphere shows procedural texture obtained by evaluating the procedural texture present on the second sphere but surface points were distorted (warped) using the procedural texture present on the first sphere.

Planar texture modeling assumes that the domain of a texture is a

plane. Modeling texture on a plane is much more tractable than modeling on curved surfaces in 3D. In this approach procedural texture is described in 2D space while leaving the third coordinate to zero. Once procedural texture is described for planar surfaces, various planar projections can be used to apply the texture on curved surfaces in 3D space (Figure 3.8) as discussed by Akenine [1], ch. 6.

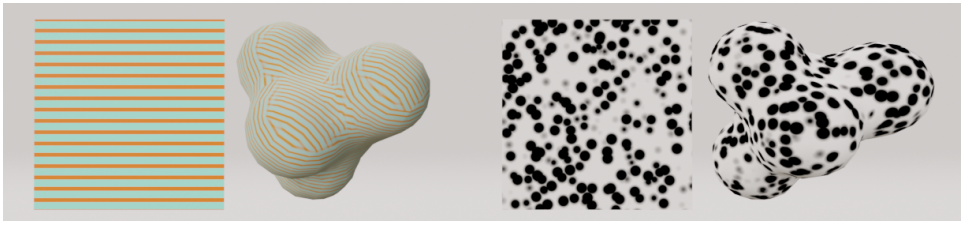


Figure 3.8: Left: Procedural line patterns are modeled on a planar surface and for more complex object geometries, planar texture can be mapped using, for example, triplanar mapping [111]. Right: Solid texture defines spheres in 3D space which are intersected by 3D object surface and visible as circles. As such, solid texture can be directly applied to arbitrary 3D geometry.

Solid texturing is extending the idea of planar texturing in 2D to 3D. Each 3D point on a 3D object surface can be used to evaluate a solid texturing model as discussed by Pharr et al., ch 10 [14]. This approach is useful since procedural solid texture can be directly applied to any geometry (Figure 3.8). On the other hand, not all texture patterns are easily defined directly in 3D.

Simulation approach enables the generation of complex texture patterns by defining a set of rules which are solved over time. An example of such simulation is reaction-diffusion discussed by Witkin et al. [112].

Inverse procedural texturing approaches automated the process of building a procedural texture model given image data. Hu et al. [113] present the approach where a procedural texture model is constructed by combining procedural masks and noise functions inside each region. Shi et al. [114] present an approach where the best fitting procedural model is selected from a library of procedural models and which parameters are fitted to best match the given image. Hu et al. [115] further build on this approach.

Generative AI. The idea is to *teach* machine learning or deep learning model about procedural modeling strategies and building blocks so that it can generate a procedural model in the form of a graph or algorithm. Guerrero et al. [98] present an approach where a procedural model (i.e., procedural texture graph) is built from scratch using the library of building blocks (e.g., nodes). This way, the authoring step is simplified but completely controllable and explainable. Further, enhancements and improvements on the

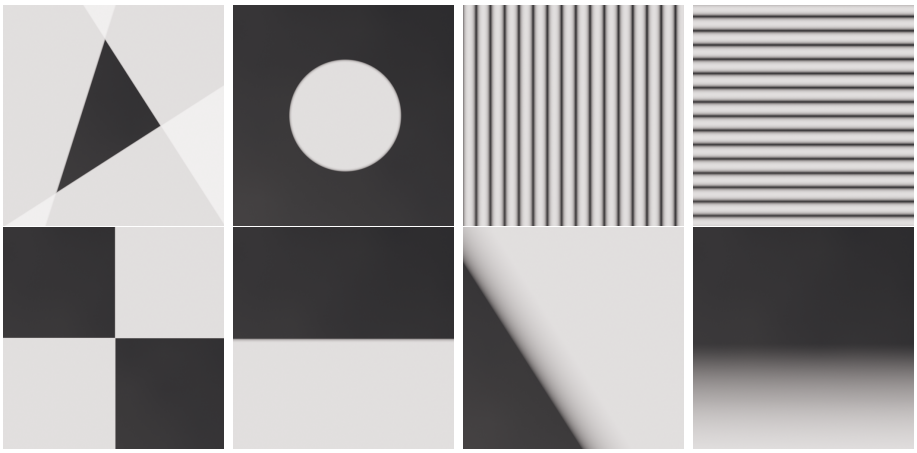


Figure 3.9: Examples of regular shapes created using procedural modeling.

resulting model can be done by an artist or computer graphics expert. It is important to note that in this approach, the machine learning model is not substituting the whole process, rather it serves as a helping tool for artists or computer graphics experts.

Building Blocks for Regular Texture Patterns

In this section, fundamental building blocks for describing regular patterns as discussed by Ebert et al. [62], ch. 2 and Vivo et al. [63] are covered. This set of methods can be seen as tools for algorithmic drawing. Many of these methods combined result in surprising complexity which is well visualized and investigated in ShaderToy [116].

Transformations such as translation, rotation and scaling are crucial building blocks for defining texture patterns (Figure 3.9).

Step function is a primitive building block which can be used as *if* statement or to produce a sharp transition from one to another type of texture. As such it can be used for building simple patterns such as polygons (e.g., triangles or cubes) (Figure 3.9).

Smooth step function represents conditional or transition similar to step, but the transition is smooth. A gradual, smooth transition (Figure 3.9) is obtained by using the cubic function, e.g., $3x^2 - 2x^3$. A smooth step function is often desired since transition can be eased in and out to avoid unpleasant or unnatural transitions present in the transition offered by a step function.

Clamp function returns value a if input value x is smaller than a, input value when input x is between a and b, finally value b if input value x is larger than b. The clamp function can be seen as conditional.

Min and max functions are closely related to the clamp function since

min and max can be expressed as clamp or clamp can be used to express min and max. Min and max functions can be seen as conditionals.

Abs function can be used to convert negative values to positive which can be seen as converting the direction of surface irregularities on the same side.

Sin and Cos functions are well-known periodic function useful for representing lines (Figure 3.9). They are closely tied to the geometry of the circle, angular measure, etc. Further, other functions can be built using the sum of sin functions of different frequencies and phases known as spectral synthesis discussed by Gardner [117].

Mod function is another important periodic function. It gives the positive reminder obtained when dividing a by b. Using the mod function as input to other functions makes them periodic too. The integer part of the ratio can be also obtained using the mod function which can be very useful for creating a spatial grid where each vertex corresponds to the integer part.

Floor and ceil functions return the largest and the smallest integer less or equal than or greater or equal than a given value.

Splines are useful for interpolating given values such as colours (e.g., colormap) or points. Often, cubic spline such as Catmull-Rom [118] is used.

Mix functions are controlled by values in range $[0, 1]$. They can exist in many forms but the main concept is they are remapping the unit interval. Examples are the gamma correction function as well as the gain and bias functions described by Ebert et al. [62], ch. 12.

Shapes such as polygons (triangles, rectangles), circles, stars, polar shapes, etc. (Figure 3.9) can be constructed with presented building blocks. Those simple shapes can be further combined into complex patterns as discussed by Vivo et al. [63].

Building blocks for Irregular Texture Patterns

To model irregular patterns, with large variety and complexity, noise functions are essential building blocks. Noise is a pseudo-random function useful for breaking the regularity and monotony of patterns. In this section, a short overview of noise functions based on the discussion in Ebert et al. [62], Dong et al. [53], Bailey et al. [119] ch. 10. and Fernandes [120] is provided.

White noise is the simplest stochastic function. It is a source of truly random numbers, uniformly distributed with no correlation between successive values (Figure 3.10). It can be truly generated only by random physical processes such as the thermal noise of analogue electronic systems.

Pseudo-random noise functions can produce a fair approximation to white noise. However, there are several properties of white noise which are not desired. For example, no repeatability, no controllability, high frequency, amount of details, etc. Therefore, the properties of desired noise functions are:

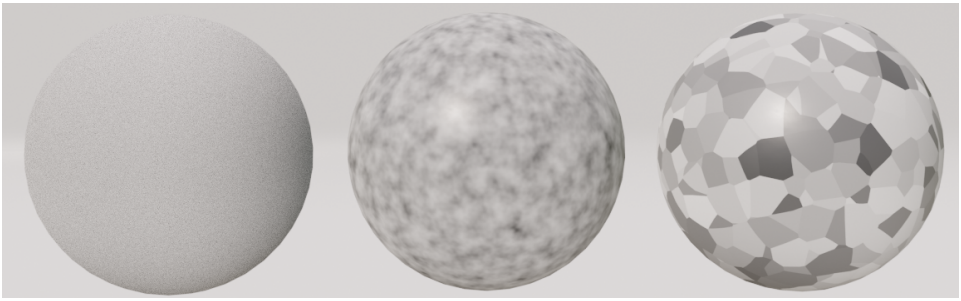


Figure 3.10: Left: white noise. Middle: Perlin noise. Right Worley (cellular) noise.

- It has to be a repeatable pseudo-random function of its inputs (e.g., the position of a surface point).
- Output of noise function must have known range, e.g., $[0, 1]$
- Noise function must be band-limited with a certain maximum frequency.
- Noise functions should not exhibit obvious periods or regular patterns.
- Noise function should be stationary (translationally invariant).
- Noise function should be isotropic (rotationally invariant).
- The Value of a noise function should change slightly if the input is changed slightly.

Lattice noises are the most popular noise types. Fundamental lattice noise function which motivated many other noises is introduced by Perlin [11] (Figure 3.10). First, pseudo-random values are uniformly distributed at the vertices of an integer lattice (coordinates of lattice vertices are integers). Second, a value at every point in 3D space (inside the lattice) is generated by the smooth interpolation of pseudo-random values on the integer lattice (interpolation is effectively performing low-pass filtering). Lattice noises may vary in how pseudo-random numbers on integer lattice points are created and interpolated. Perlin uses tabulated values and hashing to create pseudo-random values at the vertices of an integer lattice.

Value noise creates pseudo-random number (PRN) between $[-1, 1]$ at each lattice point as discussed by Perlin [121]. For a given input, the value of the noise function is computed by interpolating between these values. The key design decision of value noise depends on how interpolation inside the lattice is done. For example, linear interpolation contains obvious cell artefacts and sharp changes. On the other hand, cubic interpolation produces

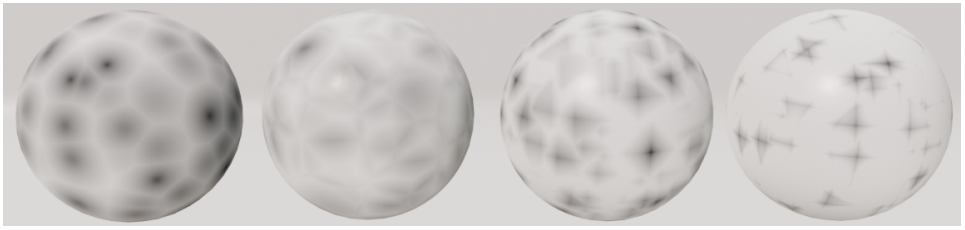


Figure 3.11: Worley cellular noise with different distance metrics: Euclidean F1, Euclidean F2, Manhattan F1, Minkowski F1.

smooth and gradual change. Many other interpolation schemes exist, for example, quadratic or cubic B-splines.

Gradient noise first assigns pseudo-random gradient vectors at each lattice point as discussed by Perlin [122]. Next, for a given input, pseudo-random gradient vectors at eight corners of a single cell are interpolated for a new value representing the noise value. An improvement is introduced by Perlin et al. [123] to mimic the swirling flow and advection.

Value-gradient noise solves the problem of gradient noise which has a value of zero on lattice points resulting in the observable grid. A simple approach is a weighted sum of values and gradient noise is returned. A more complex approach is to use cubic Hermite interpolation to combine value and gradient noise [62].

Lattice Convolution noise solves the problem of lattice noises which exhibit axis-aligned artefacts. These artefacts are due to the anisotropic nature of the interpolation schemes used for blending the pseudo-random lattice values as discussed by Ebert et al. [62], ch. 2. Lattice convolution noise is removing anisotropy and solving this problem by using discrete convolution to perform the interpolation. Lattice pseudo-random values are convolved with a radially symmetrical filter, for example Catmull-Rom filter. The convolution is simply the sum of the product of each lattice point pseudo-random value times the value of the filter function based on the distance of the input point from the lattice point.

Sparse Convolution Noises are not based on a regular lattice of pseudo-random values. Sparse convolution noise value is computed by convolving a filter function with a collection of randomly located random pseudo-random values (e.g., Poisson process). Filtering can be done using for example Catmull-Rom filter. Scattered pseudo-random values are considered sparse in contrast to white noise, thus name *spars convolution noise*. An example is spot noise discussed by Wijk et al. [124]. Due to randomly placed points with pseudo-random values, a large neighbourhood of cells must be considered for evaluation, thus this noise function is computationally expensive. However, it results in a less observable grid-like pattern as discussed by Ebert et al. [62], ch. 2.

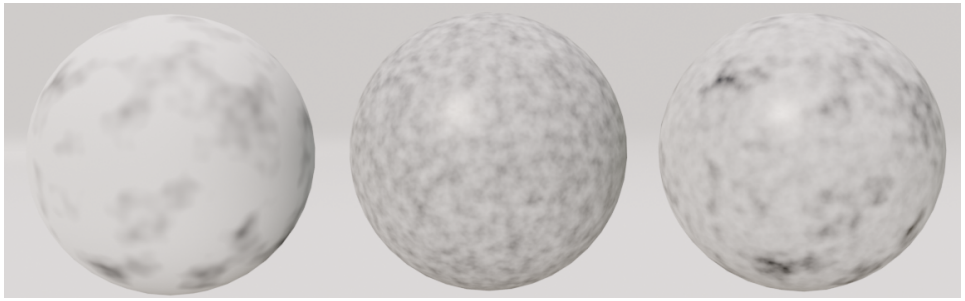


Figure 3.12: Fractal-based noise. Left: Perlin noise with small frequency (scale) and high amplitude. Middle: Perlin noise with high frequency and small amplitude. Right: Fractal noise made by adding previous two.

Cellular noise introduced by Worley [64], is based on randomly distributed discrete features (i.e., points) in 3D space. Noise value is computed by finding the distance $F(p)$ between the input point p and the feature points. Distance $F(p)$ can be computed using different metrics such as Euclidean, Minkowski, Manhattan, etc. The first closest distance between p and the feature point determines the distance $F_1(p)$. $F_1(p)$ switches from one feature point to the next along equidistant planes which are exactly the planes computed by the Voronoi diagram - a partition of space into cellular regions (Figure 3.10). Similarly, distance function $F_n(p)$ can be defined as the distance to n-th closest feature point to input point p. Different distance functions $F_i(p)$ form different patterns (Figure 3.11) and further complexity can be achieved by combining them in a certain expression, e.g., $F_1(p)+F_2(p)$. Further complexity can be achieved by combining multiple cellular basis functions in a fractal fashion. The cellular basis function is continuous and it can be computed anywhere in space. Due to its nature, it is extremely useful for describing any kind of cellular-like pattern.

Fractal-based methods are often used to create more complex patterns. The idea is to combine multiple noise functions (Figure 3.12). Perlin [11] introduced *turbulence* function which sums up noise functions, similarly as *fractal sum*, with different frequencies. Changing the frequency of the noise function affects the average size of gaps called *lacunae*. Scaling the amplitude of noise function affects *fractal dimension*. Combining noises with doubled frequency and halving the amplitude of each layer results in fBm (fractional Brownian motion) [11], [63] ch. 13.

Explicit Noise Algorithms are a class of noises which are not convenient for implicit procedural texturing models (Ebert et al. [62], ch. 2.). These methods generate all noise values all at once explicitly. To use them in an implicit procedural texturing model, values must be stored as a table or image and then used. Examples are midpoint displacement and random successive additions. Such methods can be less expensive compared to im-

plicit evaluation methods. **Fourier Spectral Synthesis** (Ebert et al. [62], ch. 2.) is an example of explicit noise generation. The idea is to generate a pseudo-random discrete frequency spectrum where power at a given frequency has distribution for the desired noise. Then discrete inverse Fourier transform (e.g., inverse FFT) is performed on the frequency domain to obtain a spatial domain representation of the noise. This approach is slower than lattice noises and not practical for procedural texturing.

3.1.2 Generation Phase

In the generation step, the procedural model is evaluated with specific parameters to produce the final texture. The realization of procedural texture depends on the procedural texture type: implicit or explicit as discussed by Ebert et al. [62] and Deguy [10].

Implicit procedural texturing models are evaluated, when needed, by answering a random query from the rendering procedure. They are suitable both for ray-tracing-based rendering as well as rasterization-based rendering since both can be seen as sampling the surface, where each sample can be seen as a query. Therefore, implicit procedural texture models are implemented in a shader which is then used during rendering. The rendering procedure provides information on shading point which is used for evaluating the procedural texture model. Only once the rendering procedure is finished, the texture will be visible on the object surface - texture is generated on the fly and no information is stored as a texture map making it memory efficient.

explicit procedural texturing models generate the complete texture pattern as a raster image for a given resolution. Therefore, storing raster images in memory is required. The generated image is then used for answering the rendering procedure query. Although it requires more memory during rendering, querying a raster image texture is faster.

In theory, both implicit and explicit models can produce the same class of textures, but in practice, it is more convenient to use one over another. Both have advantages and disadvantages and their choice depends on the application:

- Implicit models can be directly used during the rendering procedure while explicit models first have to be evaluated as an image which is then used for rendering.
- Implicit models are extremely compact when it comes to memory requirements while explicit require storing the whole pattern as an image.
- Implicit models enable render-time evaluation with parameter variation while explicit models have to be evaluated pre-render-time completely for each variation of parameters

- Implicit models ensure high quality for changing view distance or area to be covered while explicit models are restricted to the resolution of the raster image in which the model is evaluated.

- Rendering query evaluating implicit models can be slower compared to direct raster texture image lookup which is generated pre-render-time with explicit models.

- Certain complex texture patterns are simply too hard to write implicitly. This is due to the lack of spatial information which is naturally available when writing an explicit model.

3.1.3 Applications

Presented strategies and building blocks used for authoring a procedural model as well as its usage are best illustrated by the set of applications where procedural texturing is used. Concepts discussed in these specific applications can be generalized to different applications such as modeling machined surface textures.

Procedural texture modeling toolsets. Industry-standard procedural texture modeling for games, film and beyond are Substance Painter [12] and Houdini [107]. These environments combine discussed strategies and building blocks for procedural texture modeling. Although powerful, they require experience, computer graphics and artistic knowledge.

Content creation for film and game industry relies on procedural texturing for controllable and fast production of asset instances which are used for populating the 3D scene. Attractive characteristics of procedural modeling are increasing productivity, complex texture production, look consistency, animation, etc. as discussed by Deguy [10].

Synthetic image data generation for machine learning. Development of computer vision algorithms based on machine learning requires large and diversified datasets. A solution for generating required datasets lies in image synthesis. Procedural texture modeling is a crucial element for creating parametric 3D scenes which are used for diversified image synthesis as discussed by Tsirikoglou [125] and Mayer [126].

3.2 Procedural Geometry Modeling

Geometry is used to represent the shape of an object. Procedural geometry modeling is a way of generating geometry using an algorithm. Geometry can be generated in various ways using procedural modeling: generating geometry from scratch, modifying existing geometry, deforming existing geometry, appending geometry to existing geometry, combining existing geometry into new geometry, etc.

Similarly as in procedural texturing, procedural geometry modeling consist of authoring and generation step. The authoring step focuses on procedural model development. The generation step focuses on procedural model evaluation and generating instances of geometry (i.e., during rendering).

The authoring step consists of designing and developing a procedure which generates geometry. The procedure is described using code [127] or visual programming [107]. Writing a procedure which generates geometry is based on various strategies and building blocks which are combined and extended as it will be discussed in Section 3.2.1.

The generation step is the process of evaluating the procedural geometry model with a fixed set of parameters resulting in a geometry instance. The generation step depends on the type of procedural geometry model which can be implicit or explicit. In *implicit models*, the model is evaluated by rendering procedure, thus complete geometry is available after the rendering procedure has finished. In *explicit models*, the model is directly evaluated and generates the whole geometry. Implicit models, such as signed distance functions, are particularly suitable for ray-tracing-based rendering for which visibility solving, that is, intersection testing can be elegantly performed. On the other hand, explicit models are suitable both for ray-tracing-based rendering as well as rasterization-based rendering since the resulting geometry can be treated like any other input geometry. The generation step is further discussed in Section 3.2.2.

The advantage of procedural geometry modeling compared to manual geometry modeling is:

- Parameterized and thus controllable creation of a large number of geometry instances
- Automated and controllable deformation, modification or population of geometry
- Ability to create an immense amount of detail, adaptive to view distance

On the other hand, procedural geometry modeling has several downsides:

- It is hard to restrict parameters to always produce a meaningful resulting geometry instance
- Intuitive control over local geometry features is not easily obtainable since parameters often dictate the global distribution of all features.

3.2.1 Authoring Phase

Procedural modeling of geometry, similar to procedural modeling of texture, is still an art form: no recipe exists that can be reused. The authoring

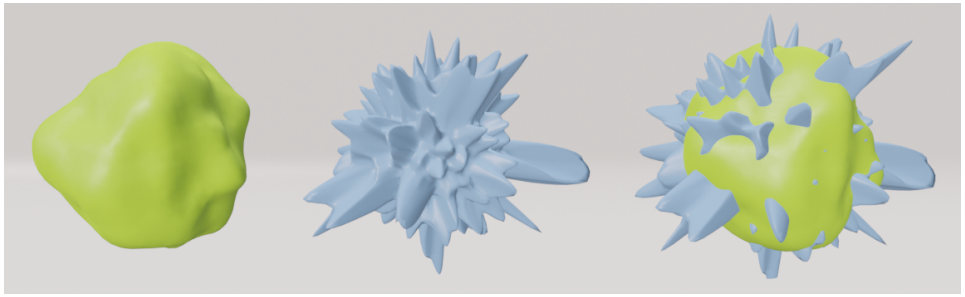


Figure 3.13: Layering concept. Two geometries are modeled separately representing different layers. The final geometry is obtained by merging the layers.

process thus relies on strategies and building blocks which are combined and extended.

Procedural Geometry Modeling Strategies

Layering is a fundamental concept which was identified that makes modeling complex shapes easier. The idea is to model complex shapes layer by layer which are then combined (Figure 3.13). This way, complex problem is simplified into smaller, simpler problems. Layers can, for example, represent different scales. Therefore, the first layer defines the coarse shape, the second layer defines smaller-scale geometry, the third even smaller scale and so on. On the other hand, layers can represent parts of geometry which are modeled using different techniques. For example, certain parts can be simulated using physically-based methods while other parts can be modeled using phenomenological methods.

Stochastic geometry is concerned with the study of random spatial patterns [128]. Therefore, stochastic geometry modeling serves as a good starting point and inspiration for building procedural geometry models. Stochastic geometry modeling systematically describes elaborate random spatial patterns using spatial point processes and random sets [129]. Combining point processes and random sets results in a particularly useful *germ-grain model* [108]. *Germs* can be seen as points generated by a point process. For example, germs can represent sampled 3D points on an object's surface or in 3D space. *Grains* can be seen as a compact close set which is placed on germs (Figure 3.14). For example, grain can represent geometrical shapes such as spheres or cubes. This concept is particularly useful since it decouples complex geometry into modeling simpler shapes (i.e., grains) and modeling their positions (i.e., germs).

Random tessellations is subdivision of 3D space into cells - non-overlapping, compact sets [128]. Point processes and random sets described in stochastic geometry modeling are finding particular usage when it comes

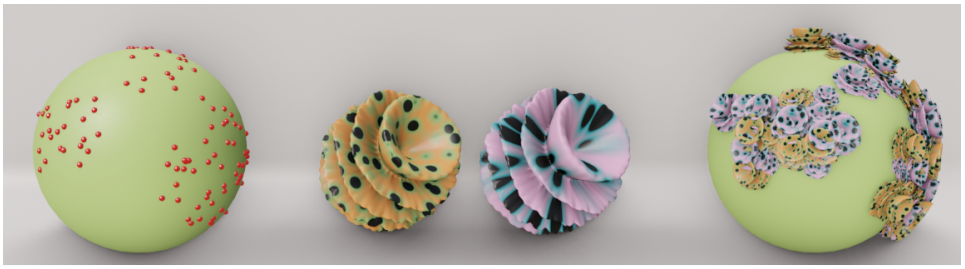


Figure 3.14: Modeling complex geometry can be decomposed into modeling simpler shapes and modeling their placement. Left: placement modeled for geometric instancing. Middle: simpler geometry modeled separately. Right: Simpler shapes instanced on sampled points.

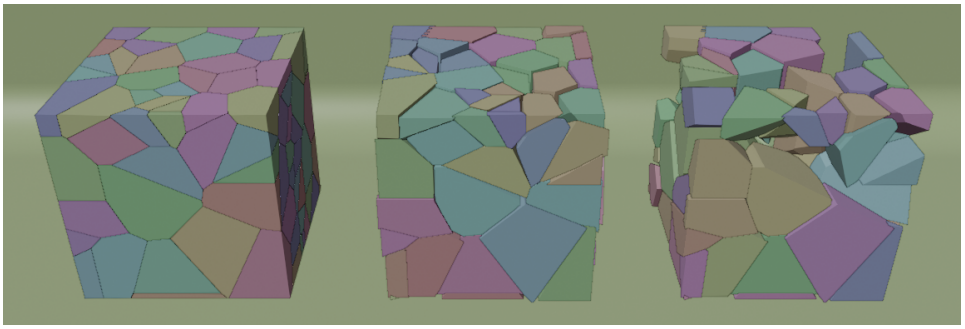


Figure 3.15: Initial cube geometry is tessellated in Voronoi cells. The resulting cells can be transformed (e.g., scaled) and randomly removed to create new shapes.

to creating random tessellations. Widely used random tessellation is Voronoi and Delaunay tessellations. Random tessellations are a useful concept in procedural geometry modeling (Figure 3.15). For example, existing geometry can be tessellated and a subset of resulting cells can be either removed or re-scaled to modify and thus introduce imperfections in the input geometry. Another example is to create fractures using tessellation cells [130].

L-system is a grammar of replacement rules discussed by Lindenmayer and Prusinkiewicz [131]. Most widely, the L-system is used for modeling biological shapes and development such as vegetation and natural phenomena but also artificial shapes and patterns. L-system grammar consists of symbols such as "F", "+", "-", etc. L-system defines a set of rules, known as production rules, that describes the replacement of a non-terminal symbol with a string of zero or more symbols. L-system is seeded with *axiom* - initial string on which production rules are applied. Each symbol defined by the L-system can be assigned geometric elements such as translations, rotations, scaling, shape (e.g., cylinders) etc. L-systems are organized into families based on their representational capabilities. The simplest is a de-

terministic, context-free L-system. Parameters and stochasticity are further added to simulate randomness. Varying degrees of context sensitivity can be used to achieve global influences across the L-system (e.g., tropism).

Geometric instancing. Composite geometrical objects can be described using a scene graph and instanced into the scene as a sub-tree of the main scene graph (Figure 3.14). Instancing can be used to implement productions of an L-system as discussed by Hart et al. [132], thus geometric instancing has representational power comparable to deterministic, context-free L-system. Instancing of scene graphs can be further described with a procedure enabling procedural geometric instancing. Procedural geometric instancing has representational power comparable to stochastic, context-fee parametric L-systems.

Fractal geometry represents a geometrically complex object, whose complexity arises from the repetition of a given form over a range of scales as discussed by Ebert et al. [62], ch 14. Fractals are the simplest complexity generators. Such an object holds the property of dilation symmetry - being invariant under a change of scale. Fractal complexity is a result of repeating the same rules or events. Non-fractal complexity is characterized by the accumulation of distinct and unrelated events over time. Fractal geometry is obtained by repeating an underlying shape - a basis function which has a certain frequency, amplitude and lacunarity (change of frequency in each step of the repeating process). Any complex, self-similar phenomena over scales can be well described with fractals. Fractals fit well into the procedural modeling paradigm since they can be described with iterative, constructive procedures. Fractals can be generated using various methods: Iterated function systems [133], L-systems, Mandelbrot sets [134] as an example of escape-time methods, diffusion-limited aggregation [135] as an example of random fractals, etc.

Generative Modelling Language, discussed by Havemann [136], is paradigm where 3D geometry is described using geometry-generating rules. Simple geometry-generating rules can be combined into complex geometry-generating rules for describing complex geometry. An example is the usage of Euler operators [137] which modify the mesh by creating or removing faces, edges or vertices according to simple rules while preserving the overall topology.

Procedurally driven physical simulation enables modeling complex behaviour of 3D objects in a 3D scene given driving forces. Unified physical simulation is described by Lesser et al. [138]. Parametric modeling of complex phenomena can be decomposed into procedural modeling of driving forces and physical simulation solving the resulting behaviour. This way, procedural modeling is only focused on generating driving forces in the form of vector fields. An example of a procedural vector field is flow noise discussed by Perlin et al. [123].

Inverse procedural modeling represents approach for creating pro-

cedural model from data [139], [140]. Given an exemplar geometry, the task is to find the set of rules which are similar to the exemplar as discussed by Bokeloh [141]. This way, building a procedural model can be simplified given enough data. Examples include inverse procedural modeling of trees [142], branching structures [143], facade layouts [144] and architectural models [145].

AI-based procedural geometry. Deep neural networks, the most popular form of AI, can be applied to 3D geometric data for parametric 3D shape modelling and animation. In this approach, geometry and parameterized rules are described using a neural network. A notable example is work done by Pearl et al. [146] where 3D shapes are mapped to a human-interpretable parameter space, allowing intuitive editing of the recovered 3D shapes from a point cloud or sketch input. Another example is work by Shechter et al. [147] where neural networks are used to inject the underlying shape geometry into the deformation parameters.

Procedural Geometry Modeling Building blocks

Transformations are fundamental for procedural geometry modeling. Transformations can be applied to the whole geometry or some of its parts (e.g., vertices). Since the work in this thesis relies on meshes, transformations can be applied on the whole mesh (all vertices) or per vertex. When applied to the whole mesh, scaling transformation is particularly useful. Scaling can be applied uniformly: equal in all scaling directions or non-uniformly: not equal in all scaling directions. When applied to each vertex, translation and rotation transformations are particularly useful. This way, a mesh can be deformed by transforming each of its vertices procedurally.

Displacement is a useful method for introducing deformation of geometrical surfaces. It can be done by offsetting vertices in a normal direction using the height amount for this vertex as introduced by Cook [148]. The height value per vertex is given by texture which can be generated using procedural models, for example, noise functions (Figure 3.16). Displacement is often combined with subdivision since for high-frequency height variation per object surface higher subdivision of geometry is required. Otherwise, the resulting geometry will lack geometrical details. This problem can be seen as aliasing in 3D as discussed by Watt et al. [149].

Texture can be seen as a function which for each surface point or point in 3D space generates a value and thus can serve as a procedural geometry modeling tool as discussed by Ebert et al. [62]. For example, texture can represent the height field which is then used for displacement (Figure 3.16). Further, texture can be used as a weighting scheme for instancing or deformation.

Deformation modifiers are a set of tools that are often available in 3D modeling programs such as Blender [150]. Various deformation mod-

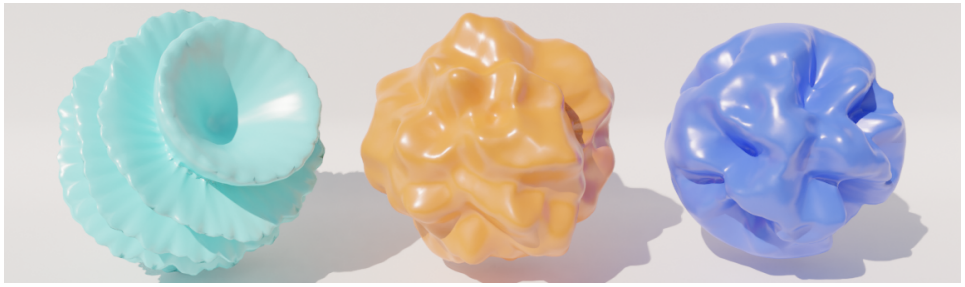


Figure 3.16: Geometries generated by displacement of spare vertices in normal direction using three different heightfields generated by procedural texture.

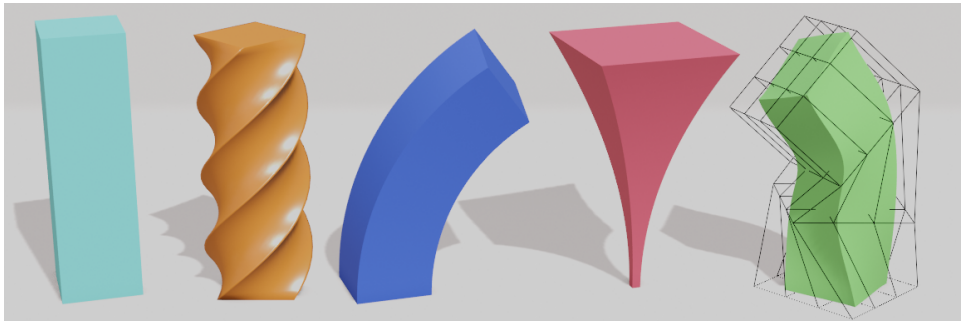


Figure 3.17: Starting geometry on the left is procedurally altered using twisting, bending, stretching and lattice deformation modifiers.

ifiers exist to deform the geometry as discussed by Watt et al. [149] and Mortenson et al. [151] (Figure 3.17). One approach is to deform the base geometry using a lattice. Lattice represents coarse geometry which is easier to control. Another approach is to deform geometry using rotation: twisting and bending. Geometry can also be deformed with scaling: taper and stretch. Warping and morphing of geometry are further useful modifiers for procedural deformation as discussed by Gomes [152].

Geometry sampling is an important tool for determining positions on the 3D object surface. Once a position is determined, instancing, random walk, deformation, displacement, etc. can be used to modify the geometry (Figure 3.18). Sampling depends on the type of geometry. Sampling is often performed on mesh faces as discussed by Portsmouth [153] and Sik [154]. This way, efficient and controllable sampling can be achieved.

Geometry walking on a surface, given a starting and optionally an ending point, creates a path. The created path can be used to instance geometry or to create parametric curves or surfaces. When the starting and ending points on the surface are given, the shortest path can be constructed. When only a starting point is given, then a random walk on the surface can

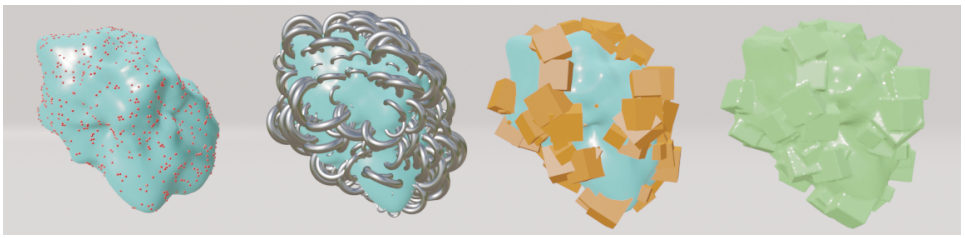


Figure 3.18: Initial geometry is sampled for positions (red points). Once samples are created, geometry can be instanced. In this case, foundational geometrical shapes such as torus and cubes are instanced. Finally, instanced geometry can be merged with base geometry to create a new shape.

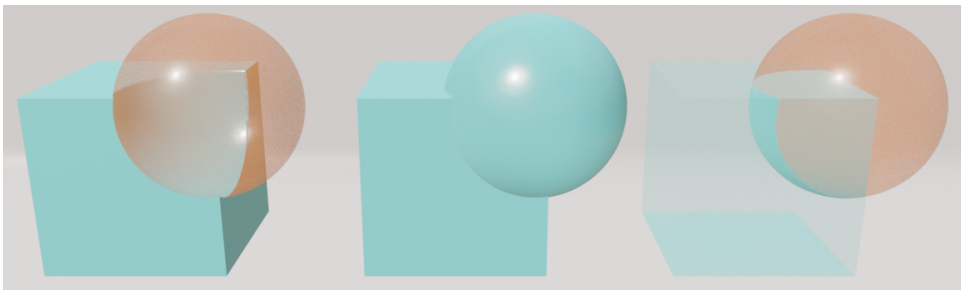


Figure 3.19: Boolean operators between cube and sphere geometry: difference, union, intersect.

be used to construct the path. The constructed path can be further displaced to introduce higher curvature in its shape. Geometric path construction relies on geodesic distances discussed by Surazhsky et al. [155].

Foundational geometrical shapes such are plane, sphere, cube, cylinder, cone, torus, etc. represent building blocks for creating more complex geometry. This can be done either by combining, modifying, instancing, etc. (Figure 3.18). The idea is that any geometrical shape, no matter how complex, can be decomposed into simpler shapes. A particular advantage of using simple shapes is that they can be completely mathematically described and thus procedurally created.

Boolean operations such as union, intersect and difference represent basic tools for combining and modifying geometry as discussed by Hoffmann [156], ch. 3. Those operations can be easily incorporated into a procedural modeling framework since they can be seen as operators between geometrical objects (Figure 3.19). An example is constructive solid geometry (CSG) [157] which uses simpler geometrical shapes such as spheres and cubes to create more complex geometrical shapes using Boolean operations.

Remeshing methods aim to improve mesh quality as discussed by Khan et al. [158]. Remeshing improves the regularity of vertices and edges as

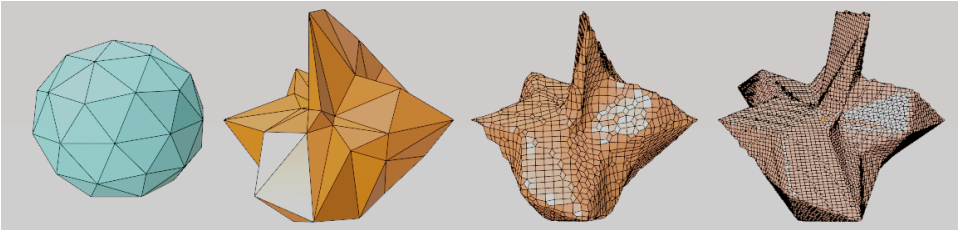


Figure 3.20: Initial sphere geometry with uniform triangles is displaced resulting in geometry with non-uniform triangles. The third and fourth geometries represent remeshed spheres with smaller and larger resolutions.

well as increases robustness. Remeshing can be used in between different geometrical operations or on the resulting geometry (Figure 3.20). This is an important operation in procedural modeling workflow since the intermediate or final geometry may contain non-uniform edges and vertices.

3.2.2 Generation Phase

In the generation phase, the procedural geometry model is evaluated with fixed parameters resulting in a specific instance of geometry. The generation phase depends on the type of procedural geometry model: implicit or explicit.

Explicit models directly generate the points which make up the shape. Thus, they are more suitable for polygonal meshes and parametric surface representations. On the other hand, *implicit models* answer a query about a particular point. Thus they are more suitable for implicit geometry representations such as signed distance functions. That said, evaluating implicit models is very natural to use during the ray-tracing-based rendering procedure since the implicit model can be queried for each ray. On the other hand, explicit models, which generate whole geometry before rendering, can be used for rendering both with rasterization-based and ray-tracing-based methods.

In principle, both implicit and explicit models can produce the same classes of geometry. However, in practice, one type is preferable to another due to various reasons such as rendering workflow, modeling workflow and a class of geometry which is to be modeled. For example, explicit models are more suitable when spatial information of the geometry is needed (e.g., for sampling).

Implicit and explicit procedural geometry models stem from two evaluation approaches: data amplification and lazy evaluation as discussed by Ebert et al. [62], ch. 11. *Data amplification* can be seen as pre-render-time geometry generation. The procedural model creates highly detailed intermediate geometry which is then passed to the rendering procedure. The

problem with this approach is the large memory cost of such a geometrical model that has to be handled. L-System is an example of a data amplification approach since it generates highly detailed geometrical representation which is then inputted to the rendering procedure. *Lazy evaluation* can be seen as render-time geometry generation. In contrast to data amplification, intermediate geometry is not generated. The synthesis of geometry is performed only when it is needed. Procedural geometric instancing is an example of a lazy evaluation approach.

As the goal of procedural geometry models is to be used as elements of 3D scenes, is important to note that efficient rendering requires efficient bounding volumes for each geometry. For geometry generated pre-render-time, bounding volume can be also precomputed before the rendering procedure. Such bounding volumes are static through the rendering process. On the other hand, a geometry which is generated during rendering requires dynamic bounding volumes as geometry is being generated.

3.2.3 Applications

How presented strategies and building blocks are combined is best illustrated through applications where procedural geometry is used. Various applications of procedural geometry are surveyed by Smelik et al. [96]. It is important to note that the concepts of those methods are usable in different applications such as manufactured surface modeling.

Procedural geometry modeling of terrains is often combining height field and displacement as discussed by Weiss et al. [111]. The height field is used to describe terrain elevation which is generated by displacement. The height field can be represented as an image or procedural texture. Procedural height fields are often generated using noise such as Perlin noise. Advanced methods for terrain modeling rely on combining procedural approaches with layered data structures, physically-based methods, AI methods and interactive methods as surveyed by Smelik et al. [159].

Procedural geometry modeling of vegetation is concerned with producing individual plant organs to complete plants and whole plant ecosystems. L-systems proved to be very powerful system for the procedural generation of vegetation. Further, procedural vegetation modeling is augmented with physical parameters such as collision detection or shadowing as well as interaction by sketching as discussed by Smelik et al. [96]. Runions further discussed space or surface sampling and then walking for path construction [160].

Procedural geometry modeling for water bodies is quite focused on modeling rivers by using height maps. One set of approaches creates a height map based on the river network as discussed by Genevaux [161]. Other sets of approaches create a river network by analyzing heightmap as discussed by Belhadj [162]. Similarly, as for other applications, interactive

modeling is introduced for guiding the procedural model.

Procedural geometry modeling for cities and roads. Procedural road generation can be constrained on terrain or city layout as discussed by Smelik [96]. Therefore, it can be seen as a certain path construction with constraints. Procedural modeling of cities often relies on a hierarchically structured model. One strategy is to first generate a broad division of city zones and then refine them. Another approach is to use L-systems for modeling as discussed by Parish et al. [163].

Procedural geometry modeling for buildings and interiors. Procedural building generation often relies on rewriting systems such as L-system, split grammar or shape grammar as discussed by Smelik et al. [96]. Interior modeling often relies on floor plan generation and furniture layout solving. Floor plan generation is often solved using grammar, subdivision, constraints-solving, etc. Furniture layouts are often solved using example-based or constraint-solving methods also discussed by Smelik et al. [96].

Procedural modeling toolset. Wide range of both specialized and general tools was developed which combine discussed strategies and building blocks for fast and effective procedural modeling. Houdini [107] represents the most general procedural modeling tool capable of creating a wide range of geometries but comes with the cost of a high learning curve. Vue [164] and Terragen [165] represent procedural toolsets specialized for modeling virtual environments containing natural elements ranging from water bodies and mountains to clouds and vegetation. SpeedTree [166] and XFrog [167] are specialized tools used for the procedural modeling of trees and vegetation.

Part II

Image Synthesis and Procedural Modeling

Chapter 4

Image Synthesis for Surface Inspection Planning

Surface inspection development requires large amounts of image data representing the inspected product surface. The image data should contain both the ideal surface and the defective surface that can appear during production. Although image synthesis comes as a natural solution to this problem, its application is not straightforward in automated surface inspection environments. The reason for that is a lot of manual work that should be done for creating defects, simulating the inspection environment, and setting up the acquisition system for validation of simulation. To address these issues, we present a novel pipeline that automatizes surface defect creation, provides realistic rendering in a predefined inspection environment setup, and an acquisition system that enables comparison with the real images. The pipeline creates geometry-imprinted defects which combined with physically based rendering methods enable realistic light response for different light and camera positions during image synthesis. Finally, synthesized images can be compared with the real image taken in the same setup enabling verification. Also, synthesized images enable the visualization of visible surfaces and defects for a given inspection plan.

The presented pipeline wouldn't be possible without the important contributions made by two students: Doria Šarić and Siddhartha Dutta. Methods and results regarding *ErrorSmith* (Sections 4.4.1 and 4.5.1) were researched, developed and described by Doria Šarić. Methods and results regarding *Acquisition system* (Sections 4.4.3 and 4.5.3) were researched, developed and described by Siddhartha Dutta.

4.1 Introduction

Industry 4.0 introduced the concepts of smart factories and smart manufacturing which are oriented towards automated production with high cus-

tomization capabilities. Automated production further requires automated inspection of the products. Due to its high availability and applicability, visual inspection is a frequent inspection method [6]. Therefore, automated visual inspection, such as surface inspection, can be considered as an important part of quality assurance in the Industry 4.0 concept. Even though the inspection process itself is automated, development of one such inspection system requires an expert approach. For that purpose, research efforts have been focused on development of automated inspection planning tools, which can be used as an aid to experts developing surface inspection systems [6], [4]. The main goal of such planning tools is to find the optimal setup of acquisition hardware (camera and illumination) that ensure complete surface coverage of the product.

Good acquisition setup and development of robust image processing algorithms for defect detection are two highly interdependent parts of inspection system design. Acquisition hardware is placed and calibrated to maximize the visibility of a defect, and the image processing algorithms are developed for images made with that particular acquisition setup. Therefore, development of automated surface inspection systems relies on a large amount of representative product image data. More specifically, image data containing as many defects as possible. Due to variety of reasons (frequency of defect occurrence, small production batches, etc.), the amount of available defects is frequently limited, which can cause problems in the development of robust detection algorithms. For that purpose synthetically generated datasets are needed. Those datasets would contain images of the product augmented with defects with a wide range of shapes and positions over the surface as they would appear in the real-world cases only after longer production periods.

Computer generated imagery (CGI) comes as a logical solution to the image synthesis problem, however, its utilization is not straightforward in the automated environments since it requires a lot of manual work done by experts. In this work, we address those problems by offering a clear pipeline for realistic image synthesis of an object with both ideal and defective surface, as well as comparison to the imagery of physical objects for verification of the results. Synthesized images offer a way to visualise which parts of the surface and defects are visible for given inspection plan (i.e., camera and light positions). As it can be seen in Fig. 4.1, the pipeline consists of three modules. The first module is responsible for defect creation directly on object geometry. The second module performs physically based rendering of the object. The third module allows real image acquisition and therefore verification of the synthesized image.

The presented pipeline enables controllable image synthesis. To explain, the pipeline is founded on computer graphics simulation and rendering which are completely controllable. Therefore, the context which is used for generating images is completely controllable.

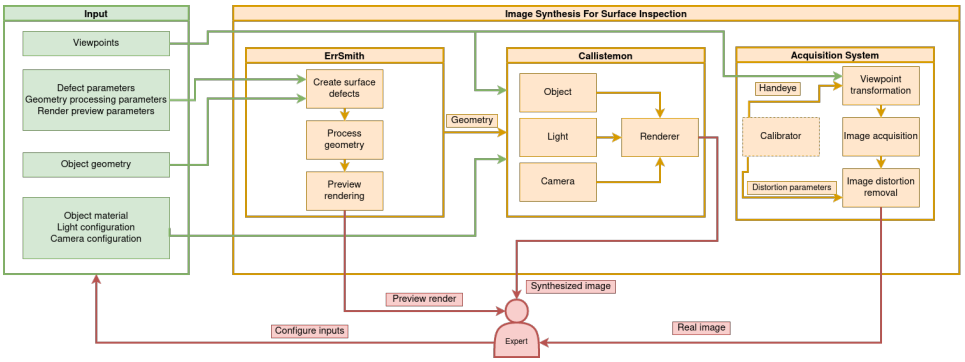


Figure 4.1: Pipeline overview. Input data and its flow is represented in green color. Core pipeline and its modules are represented in orange color. Module outputs and expert interaction is represented in red color.

4.2 Related Work and State of the Art

Gospodnetic, Mosbach, Rauhut and Hagen [6] defined fundamental requirements for surface inspection planning pipelines and gave an example of a semi-automated visual inspection planning pipeline. The presented pipeline uses the geometry of the object for the purpose of viewpoint placement. The problem of data synthesis as a means to predict the inspection outcome has not been tackled, but the importance of such work has been highlighted. The pipeline also introduced a verification system, used for coverage comparison, but the method used for correct positioning of the camera was not discussed.

Deficiency of defect data motivated defect image augmentation (Shorten and Khoshgoftaar [168]). Standard image transformations are used to produce more data, which in some respects becomes redundant or repetitive, but makes the detection robust to defect position, size, etc. Autoencoder and GAN-based solutions can infill defect-free images with synthesized defects learned on real defect images (Tang, Yang, Xiong and Yan [169]). However, the learning process is unstable without larger amounts of data and requires a confident validation for generated images. Even so, it could only be applied to simple geometries in ideal inspection lightning conditions.

As an alternative, Merillou, Dischler and Ghazanfarpour [170] propose a hybrid approach to effectively render surface scratches. They couple bidirectional reflectance distribution functions (BRDFs) with texture mapping. However, it involves taking measurements of real scratches and the profile curve of the defect before mapping it to the model. Desbenoit, Galin and Akkouche [171] built an atlas of template fracture models imitating real cracks on various materials. The fracture model is automatically mapped onto the 3D model and carved into object surface using the Boolean difference operator. Although lightweight, it is still time consuming.

Haindl and Filip [172] provided taxonomy of the reflectance models, highlighting the scale of the observation as an important aspect. BRDF represents an efficient model for homogeneous surfaces (i.e., surfaces seen from a larger distance). Ngan, Durand and Matusik [47] evaluated analytical BRDFs in terms of their ability to fit measured bidirectional distribution function data. They concluded that microfacet based models (Walter, Marschner, Li and Torrance [9]) provide a faithful, physically based appearance.

Mohammadikaji [4] and Reiner [173] tackled with realistic image synthesis for machine vision inspection systems. Mohammadikaji coupled ray tracing with Fourier optics methods for introducing wave optics effects. While it produces good results for laser based illumination, in case of white light illumination Dong, Walter, Marschner and Greenberg [174] concluded that same result can be achieved using geometric optics models which are widely available in existing physically based rendering engines. Therefore, this approach lacks generality. Reiner [173] proposed more general simulation approach but lacks clear pipeline.

Evaluation of realism in photorealistic rendering has been oriented towards appearance and was used to assess the subjective experience of a viewer (Kolivand, Sunar, Kakh, Al-Rousan and Ismail [175], Greenberg et al. [176]). The reason likely lies in the fact that recreating the same environment is a challenging process. Recently, Stets et al. [177] introduced a reassembly and multimodal digitization pipeline containing a controlled environment to precisely capture a given scene, obtaining almost pixel-precision when comparing simulated and captured images.

Obtaining correct camera setup involves camera calibration and hand-eye calibration. Camera calibration determines relative orientation of the camera to a calibration object and also its internal parameters, while hand-eye calibration determines the rigid transformation from robot end-effector to camera coordinate system. For hand-eye calibration, Tsai and Lenz [178] suggested a star-like viewpoint pattern while acquiring checkerboard images. Pachtrachai, Allan, Pawar, Hailes and Stoyanov [179] suggested calibration without markers. The downside of this approach is the lack of accuracy. Antonello, Gobbi, Michieletto, Ghidoni and Menegatti [180] and Wang, Lu, Hu and Li [181] proposed the approach in which the robot automatically takes images of the calibration object and performs hand-eye calibration. Recently, Lyngby, Matthiassen, Frisvad, Dahl and Aanæs [182] used hand-eye calibration algorithm developed by Liang and Mao [183] for accurately aligning the camera position and orientation of the robot to measure the BRDF. None of those approaches discuss cameras with a narrow field of view, which is essential in the most surface inspection scenarios.

4.3 Requirements

CGI techniques such as [28], [184], [14], [35] are commonly used for generation of photo-realistic images and are capable of producing remarkable results. However, their practical use in the field of surface inspection is minimal [4], [173]. We assume that the explanation lies in the following reasons. Firstly, defects can occur in a wide range of shapes and positions over the surface introducing uncertainty which must be dealt with. Secondly, CGI techniques are developed with the goal of providing high flexibility and freedom for artists. Artists then manually create scene elements such as objects, materials, lights etc. This makes the direct usage of such techniques not applicable for environments which aim towards automation (e.g., surface inspection development). Finally, confidently using synthesized images for representing the real physical objects requires an acquisition system followed by verification methods that can prove the quality of the object properties in the synthesized image. Guided by the aforementioned reasons, we define three main requirements which should be satisfied by a defect synthesis pipeline:

R1 Must be capable of producing defects of various shapes and positions over the surface of the object.

R2 Must produce realistic images of the object.

R3 Must compare the synthesized image with a corresponding image acquired in the real world on the spot.

4.4 Methods

As mentioned in Section 4.1, the presented pipeline consists of three main modules shown in Fig. 4.1:

- *ErrSmith*, discussed in section 4.4.1, creates defects directly on the 3D model of the object based on the provided defect parameters.
- *Callistemon*, discussed in section 4.4.2, incorporates the object material and geometry as well as light and camera properties used in surface inspection to reconstruct a 3D scene resembling the surface inspection environment. The 3D scene is then rendered from different camera viewpoints.
- *Acquisition system*, discussed in section 4.4.3, incorporates a semi automatic hand-eye calibration and camera calibration as well as automatic image acquisition of the real images taken from the same viewpoint as in Callistemon. Images acquired using the acquisition system enable verification of the synthesised images.

4.4.1 ErrorSmith

Computer graphics artists often spend a lot of time on each 3D model, sculpting surface irregularities and introducing color modification to make the object seem more real. To automate defect creation, we developed ErrSmith, a module which can automatically make geometrical surface defects on any given 3D model based on a provided set of parameters. For preview purposes, it is possible to render images of the object containing virtual defects. ErrSmith consists of defect creation, geometry processing, and preview rendering (see Fig. 4.1). It can be easily applied or adapted to various conditions and use cases.

Defect Creation

For the purpose of this work, defects are considered to be irregular indentations of varying sizes on the surface of the object (i.e., dents and scratches). Therefore, first a *defect tool* is made. It is a smaller 3D object manually modelled to resemble a 3D negative of defect representation on the surface, as can be seen in Fig. 4.2. The tool is then partly imprinted into surface of a given 3D model. Tools can be smooth and regular or quite complex with rough surfaces. The variety and complexity of defect shapes depends on the variety and complexity of the defect tools.

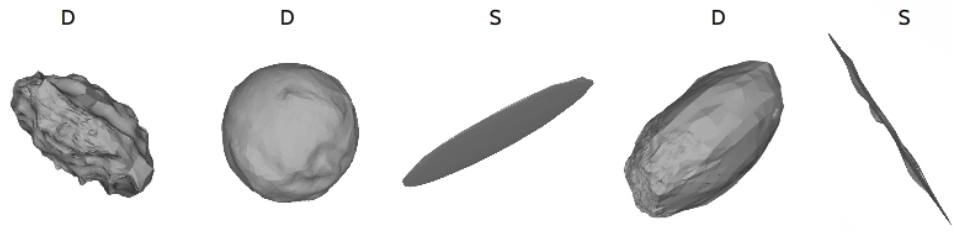


Figure 4.2: Examples of some defect tool models. The ones with the label **D** are creating dents, while the very thin ones below **S** labels are used for scratches (sharp tools imitating a saw or a blade).

The input parameters for ErrSmith define defect types, number of defects (per type or in total), and defect sizes. All of these parameters can be set to fixed values or defined as a range of values from which the value should be sampled.
The following steps are repeated **for each defect** that should be made on the object:

1. Select a random defect tool belonging to the tool collection of the chosen defect type

2. Randomly translate the defect tool anywhere on the grid

3. Scale the tool with defect size parameter

4. Randomly rotate the defect tool

5. Translate the origin of the tool to an arbitrary point on the closest face of the object

6. Apply the Boolean difference with the object mesh to create a geometrical surface defect

Geometry Processing

After creating defects, further geometry manipulations can enhance the look of rendered images. Some simple-geometry models require processing before being rendered. For example, we could add some noise to face normals to simulate a rough surface. With simple geometries, all the faces first need to be subdivided by a certain factor. Otherwise, the model will have an insufficient number of normals to simulate surface roughness. In addition to that, edge roughness can be achieved by selectively adding some noise to vertex normals.

Preview Rendering

Lightweight, preview rendering developed using Pyrender, offer a quick way to visualise ErrSmith results. Depending on the rendering parameters (camera setup, lights, viewpoints, number of images), a preview is rendered. For one given viewpoint, the preview includes a synthesized image of the defective object and a capture of the defect map. Defect map is a 3D model of that same object, with defective areas colored white and the rest in black. The captures of the defect map mark exact defect positions on the virtual image. An example of a preview can be seen in Fig. 4.3. If the rendering parameters were odd and the defects were hardly visible on the synthesized image, one could check the binary image to notice them. Additionally, the binary 2D images of the defect map could be used as the defect labels for the synthesized images in deep learning applications.

4.4.2 Callistemon

Once the defects are introduced on the object geometry, synthesis of realistic images, similar to those acquired by a visual surface inspection system, is done using Callistemon. Callistemon serves as a connection between the surface inspection planning data and a physically based rendering engine. Therefore, it offers a way of visualising defected geometry with physically based material in an simulated environment. Surface inspection planning data consists of an object geometry and viewpoints which define camera

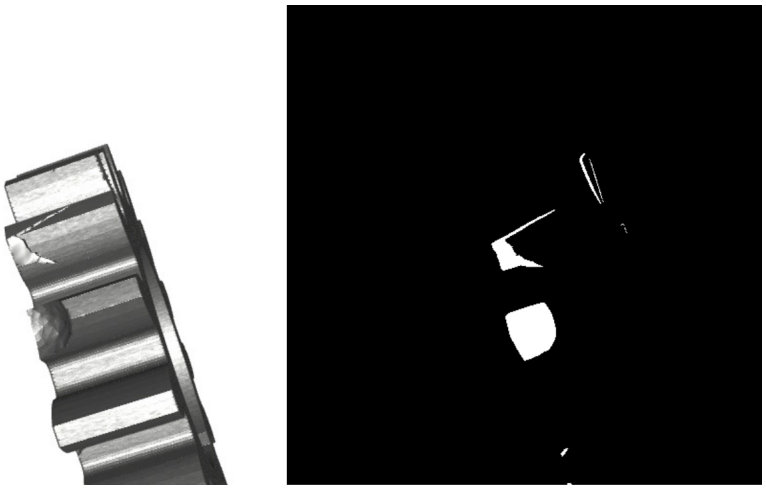


Figure 4.3: Preview of defects created on the model of a gear. Virtual image with the defects (left) and the corresponding defect map capture (on the right).

placement and orientation, relative to the given object. A viewpoint is defined as a $9D$ vector $\mathbf{v} = [\mathbf{p}, \mathbf{d}, \mathbf{u}]$, $\mathbf{p}, \mathbf{d}, \mathbf{u} \in \mathbb{R}^3$. $\mathbf{p}$ represents a point in space where the camera is positioned, $\mathbf{d}$ is the direction of the optical axis of the camera and $\mathbf{u}$ represents orientation of the camera around the optical axis. In order to synthesize an image from a given viewpoint, a digital representation of the inspection environment (scene) is required. The scene consists of a light source, a camera and an object, each of which is described by a set of inputs. Required inputs are object mesh and viewpoints. Optional inputs are object material, light source geometry and material, environment parameters and camera parameters. To systematize the rendering process optional inputs have default values, but depending on application can be adjusted.

Object, Camera and Light Parameters

To simulate the appearance of the object during inspection, its geometry and material specifications are required. The geometry should be a realistic mesh representation of the object. Material specification is separated into reflectance properties and spatial variation details (i.e., texture). Realistic reflectance properties are described using physically based BRDFs available in Callistemon's rendering engine as Open Shading Language (OSL) closures [185]. User can choose the BRDF by choosing the OSL closure and specifying its parameters. Those BRDFs are resulting in perfectly smooth surface as viewed from a large distance. Therefore, they contain no spatial variation. Spatial variation in synthetic images is a result of color and surface roughness

variation and is specified using procedural texturing. The user can describe the variation using OSL. Description accounts for variation which is much smaller than the size of the object, but much larger than the light wavelength [28]. Therefore, smaller scale of variation should be defined using BRDFs, while the larger scale of variation should be present in object geometry.

Simulating inspection environment requires that light source can be positioned relative to the camera or relative to the object. Defining the light relative to the camera will change its position in the scene for each new camera position. Otherwise, if the light position is defined relative to the object, its position $\mathbf{p} \in \mathbb{R}^3$ must be provided in object coordinate space and will remain the same for all the viewpoints. Callistemon supports physical (area) lights which are prerequisite for physically based simulation. Physical lights have certain shape and emit light from the surface of the shape. Therefore, physical light requires geometry and material specification. Light geometry should match the geometry of the real light because its area greatly influences the total amount of light in the scene. Emission material properties should be defined by the user using OSL (e.g., amount and color of light). The camera is simulated using a pinhole camera model and a set of parameters describing the real camera. The parameters include resolution, film dimension, pixel size and focal length.

4.4.3 Acquisition System

To enable validation of the synthesized images, it is necessary to obtain the real (ground truth) images in the same physical environment. Defining the environment for a surface inspection validation system is not as complex as for real life scenes (e.g., outdoor) since inspection system requires precise and consistent imaging conditions. Image acquisition conditions in an inspection system are determined by object placement, one or more cameras and one or more illumination devices. Object placement process determines the way in which the object is placed into the inspection system and manipulated during the inspection (e.g., manual placement, conveyor belt, turntable, manipulator device, etc.). Camera is always positioned relative to the object and represented by a single viewpoint from which an inspection image is acquired. Illumination devices can be relative to a camera or fixed relative to the object. Fixed illumination devices will have the same position regardless of the viewpoint which is used to acquire an image and can be used with more then one viewpoint, while relative illumination devices depend on the camera position and are typically only used by that single viewpoint.

An acquisition system provides a possibility to compare synthetic images to a precisely determined environment in an automated, consistent and reproducible manner. For that purpose we assume that the positioning system in an actual inspection system will always place the object at the same position. With such assumption it is possible to precisely define the position

of the object within the acquisition system. The camera is attached to a manipulator device, thus enabling both positioning precision and repeatability. The illumination device position can either be fixed or attached to the same manipulator device as the camera. Finally, the system is given a list of camera viewpoints defined in Section 4.4.2, which are reached using the manipulator device.

Synthetic and real images are created independently, therefore, it is not possible to rely on any kind of markers, like it has been done by Stets et al. [177]. Instead, the precise positioning of the acquisition hardware within the scene requires the application of calibration techniques. The manipulator device carrying the camera is 6 degrees of freedom robotic arm. The camera is attached to its end-effector, but the coordinate systems of the robotic arm and the camera do not coincide. Therefore, moving a camera to a precise position using a robotic arm requires hand-eye transformation describing the transformation between the end-effector coordinate system and camera coordinate system [178]. Camera is fix-mounted to the end-effector, therefore the hand-eye transformation does not change during the acquisition process, but should be recalculated with any change in the acquisition system (e.g., focal length of the existing camera). Further, it should be recalculated regularly to account for the wear and tear effects.

We introduce an semi-automatic calibration approach. First, for intrinsic calibration, images with varying viewing angles of the checkerboard which is fixed in the scene are automatically acquired. Narrow field of view at the focusing distance and large aperture cause shallow depth of field, therefore some acquired images above the pattern can not capture the checkerboard clearly. As a solution, the checkerboard is fixed in the robot space. Various boundary coordinate ranges within the robotic coordinate system are defined. The range consists of the edge points from where the full checkerboard is in view to the points where the checkerboard corners are no longer in the view. Spanning the robot hand through these ranges allows to capture different views of the calibration object. For an accurate calibration, more than 3 boundary coordinates are specified. The system requires the number of images to be acquired for the calibration, which are then taken by the robot hand striding over boundary coordinates. Calibration is sensitive to exposure which affects the detection of corners. Therefore, images without all corners detected are automatically rejected by the system. The same images are then used for hand-eye calibration. For each image the transformation of the calibration pattern to eye can be estimated from 3D-2D correspondences using Lavenberg-Marquardt optimization as done by Antonello, Gobbi, Michieletto, Ghidoni and Menegatti [180]. This approach is used for calculating the hand-eye transformation. Since the robot knows the transformation from base to end-effector, the hand-eye transformation can be calculated using the calibration method provided by Tsai and Lenz [178]. Schematic representation of the manipulator is presented in Fig. 4.4. Af-

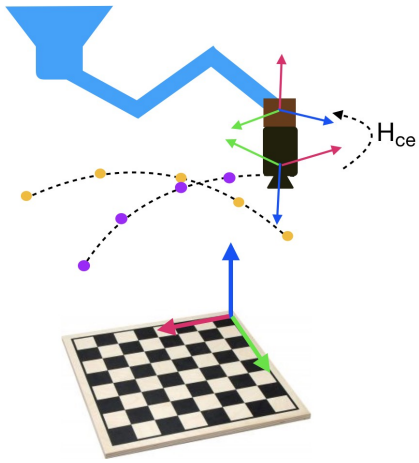


Figure 4.4: Schematic representation of how the camera is moved by the robot end-effector while taking images of the checkerboard at different positions. The yellow and purple dots show the positions calculated by the acquisition system at runtime from the given boundary coordinate ranges. H_{ce} is the resulting matrix for camera coordinate system to end-effector coordinate system transformation.

ter finding the hand-eye transformation, any point in camera space can be converted to end-effector space using:

$$[\, X_e \; Y_e \; Z_e \,]^T = H_{ce}[\, X_c \; Y_c \; Z_c \,]^T \tag{4.1}$$

4.5 Results

4.5.1 Errsmith

ErrSmith automatizes the process of modeling surface defects. There are two main types of surface defects applicable in it - scratches and dents. For each type of defect, a collection of custom defect tool models was made (both from the sphere model). All the tools are of uniform size and made with Blender, which has advanced sculpting features widely used among computer graphics artists. The examples of defect tools in Fig. 4.2 were specifically used on the gear model in Fig. 4.5 to obtain artificial defects which can be seen in the Fig. 4.8. On the central image, we can see how the first defect tool damaged a gear tooth. When running ErrSmith module, defined parameters were specified as in Table 4.1.

Defect creation	defect type	['scratch', 'dent']
	defect size	[(0.1, 0.35), (0.2, 0.4)]
	number of defects	[(1, 8), (2, 5)]
	defect coordinates	-
Geometry processing	face normals noise	Gauss(0, 0.001)
	vertex noise	-
	face subdivision	2

Table 4.1: ErrSmith parameter values for the images in Fig. 4.8.

This assumes ErrSmith will make between 1 and 8 scratches and 2 to 5 dents. Scratches will have arbitrary size from 0.1 to 0.35. Dents will be large between 0.2 and 0.4. Gaussian noise will be added to the face normals and all the faces will previously be subdivided by factor 2. There is an option to specify the number and size of defects in total (instead of separately per each defect type). For some specific use cases, the precise defect locations can be defined. The user can also specify the path to defect tool models, which enables the user to directly influence the defect appearance by modeling his own tools.

The outputs from ErrSmith, regardless of whether the images are being rendered for the preview, involve: (1) an STL file containing the defective object geometry, (2) an OBJ file containing a defect map in 3D, and (3) a JSON file containing metadata. Metadata includes all the parameter values (either default or defined by the user) for defect creation and geometry processing, paths to exact defect tools with exact defect locations they were applied to, and the amount of time it took to produce each defect.

4.5.2 Callistemon

Callistemon has been used to recreate surface inspection environment and render an object within it from the provided camera viewpoints. The scene consisted of the metal gear object and the physical light source (see Figures 4.5 and 4.6).

Object geometry was provided with the real object. Material property was configured using metal-specific physically based microfacet model (i.e., *metalBRDF*) provided in Appleseed [184]. Roughness value was set to 0.25. Anisotropy amount was set to 0.7. Fresnel normal incidence color was set to (196, 199, 199) sRGB. Spatial variation was introduced using procedural texture in OSL defined using:

$$v(x, y) = \sqrt{(x + c)^2 + (y + c)^2} \mod d \tag{4.2}$$

Where $c = S(r \mod 1.0)$, $r = \sqrt{x^2 + y^2}$. $S(\cdot)$ represents smoothstep function, $d \in \mathbb{R}$ is used for controlling pattern density, x and y are coordinates of the shading point $P \in \mathbb{R}^3$. Value of the function is multiplied with the calculated color of metal material. The resulting render is in Fig. 4.6.

Camera and light source properties were determined based on the acquisition system described in Section 4.5.3. The light source geometry was created using Blender (see Fig. 4.5). Light source material was set to diffuse emissive distribution function. Light source position is defined relative to the camera. Callistemon allows configuration of the environment light color and intensity. As the environment influence is negligible in the real setup, black, non-emitting environment is used in the simulation. Callistemon uses Appleseed, an open-source, physically based global illumination rendering

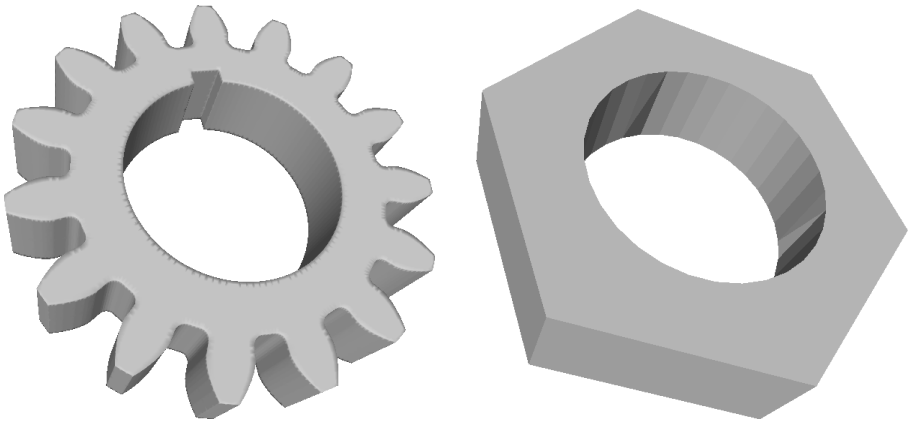


Figure 4.5: Geometry of the object (left) and light (right) used in our surface inspection environment.

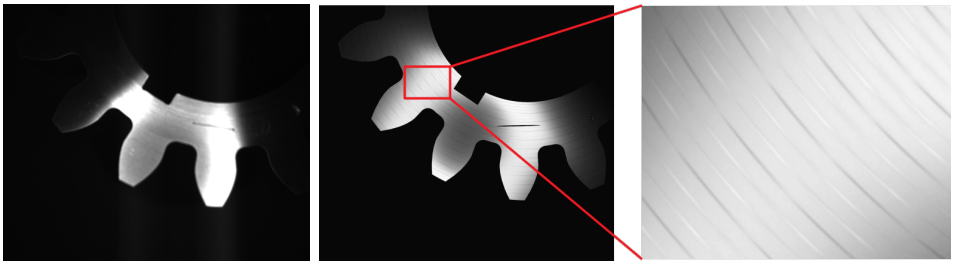


Figure 4.6: Top: real image acquired by acquisition system. Bottom: rendered image with metal material and texture which is also magnified on the right. Note the defective geometry which was introduced using ErrSmith.

engine. Modern light transport technique, unidirectional path tracing, was used along with five passes and adaptive pixel sampler.

Binary mask (see Fig. 4.7) of the object was created without any additional input. First, constant color was applied to the object. Next, environment light was enabled. These steps allow for a clear boundary between the object and the environment. Finally, using OpenCV [186] binary thresholding was performed on the rendered image. Due to discarded shading information, parts of the object that are not seen in realistic render are seen in binary image (e.g., inner side of the gear object).

Defective geometry created by ErrSmith was used in Callistemon (see Fig. 4.8). Beside defects, rendered images contain metal material as described above.

4.5.3 Acquisition System

For the purpose of this work, both the camera and the illumination device have been mounted on a UR3 robot. The camera was Prosilica GC 2450

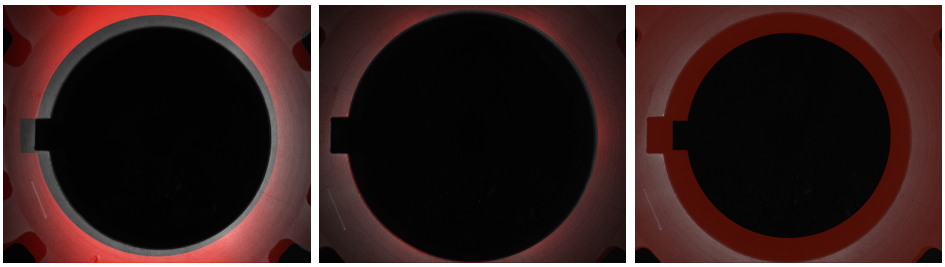


Figure 4.7: Left: overlap of rendered (red object) and real image without calibration. Middle: overlap of rendered (red object) and real image with calibration. Right: overlap of binary mask (red object) and real image with calibration.

with a Sony ICX625 CCD sensor - an industrial, grayscale camera with mounted Kowa 12mm lens. The sensor has resolution of 2448 × 2050 pixels with 3.45μm × 3.45μm pixel dimensions. The camera had a fully opened fixed aperture and focusing distance set to 146 mm. The illumination device is a DCM Sistemes ALU1006A-630C ring light mounted around the camera lens.

System Configuration

A checkerboard pattern was used to perform the hand-eye calibration. Due to shallow depth of field present in the camera setup, checkerboard with 11 × 9 squares sized 6.6 mm was used. Calibration accuracy is limited to the accuracy of the printed checkerboard, while an error by a millimeter can produce large pixel deviation in the image. As discussed in Section 4.4.3 it is required to define boundary coordinates for the robotic arm. To determine each boundary range the UR3 teach-in pendant was used. Because the boundary coordinates are manually inputted to the system, the target reachability is satisfied. To achieve low error, a number of different viewpoints were used while striding over boundary coordinates. 9 different boundary coordinates are calculated using the UR3 teach-in pendant and a total of 80 images from different viewpoints were requested to be captured. Based on that the system calculates 80 different positions from which images will be captured. Each of these positions is then randomly mixed and the robot takes one image for each of these positions. To ensure collision avoidance, the robot was made to return to a safe position before going to take the next position. To detect corners automatically OpenCV [186] and ViSP [187] libraries were used. Images without all corners detected corners are automatically rejected by the system. The rejection rate is lower when the images are of lesser resolution compared to 100% resolution. This is due to the fixed size of corner detection filters in OpenCV, where the convolution size is too small to detect corners in high-resolution images.

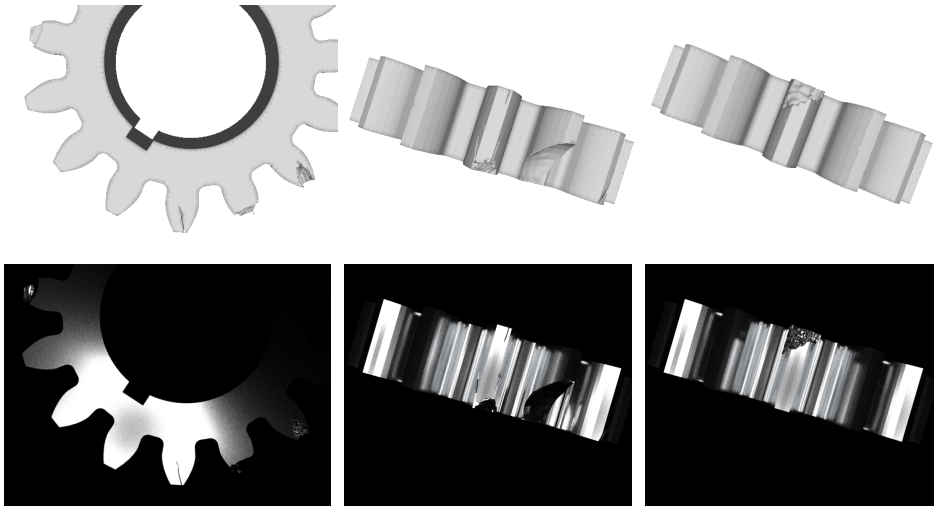


Figure 4.8: Top row: defective geometry created using ErrSmith. Bottom row: synthesized images using Callistemon. Images show visibility of surfaces and defects for particular light and camera setup. Some scratches are deep therefore light is trapped. Dents show complex light response due to complex internal structure.

4.5.4 Intrinsic and Hand-eye Calibration

The implementation of the intrinsic and hand-eye calibration has been provided by ViSP [187]. The final output of the process is a 4×4 hand-eye matrix H_{ce}, 3×3 calibration matrix K, and the radial distortion parameter k_1. These details are used by the acquisition system. k_1 is used for undistorting the images. To validate the hand-eye transformation, known 3D points in robot space are back-projected into the 2D image captured by the camera. In a robust hand-eye estimate, the back-projected 3D points coincide with the seen 2D points in the image.

4.6 Discussion

The resulting images in Fig. 4.8 show us how using just 8 defect tool models in total can produce a complex variety of surface irregularities. Since the tool is partly carved into the object, there is a great number of possibilities even for one defect tool, depending on how the tool is positioned, scaled, or rotated. Applying Boolean operator with complex defect tools allowed us to introduce more complexity and diversity into defects, while remaining computationally inexpensive. This shows that our pipeline satisfies the requirement **R1**. However, some faster and optimized Blender-external resources should be used for these operations in the future. Introducing more

defect types would enable new use cases (e.g., color modifications imitating stains). Geometry imprinted defects allow using arbitrary material during image synthesis, since defects are part of the mesh and not the material. Also, this way defects behave correctly under arbitrary light and camera positions. Additionally, there is a space for improvement of defect tool models by experimenting with different modelling techniques or other geometries. Applying physically based material and texture enabled more realistic appearance of the synthesised object as it can be seen in Fig. 4.6. Therefore, requirement **R2** is satisfied. Combination of physically based material and modern light transport results in accurate reflections and realistic appearance. Also, due to roughness, certain parts of the object are visible more clearly while other parts are not visible at all (Fig. 4.8). This is important for surface inspection tasks, because viewpoint coverage in this case can be lower than coverage estimated purely using the geometry. Clearer contrast can be achieved using advanced light distribution description such as Illuminating Engineering Society (IES), also proposed by Reiner [173]. By applying texture, our goal was to reproduce the brushing pattern introduced by industrial process. The texture is an important realism property, therefore our aim is to investigate different texture synthesis methods. Acquisition system enabled comparison of synthesized and real image satisfying requirement **R3**. As it can be seen in Fig. 4.7, without hand-eye calibration a miss-match in the overlap of the object in the rendered and the real image is present. This mismatch has been removed using the hand-eye calibration (see Fig. 4.7). Matching also depends on the distortion parameters of the camera lens. Currently, calibration system can only model radial distortion and not tangential distortion. Effects of tangential distortion can be seen in the lower right half of the image. The effect of tangential distortion increases as we move away from image center. Modeling tangential distortion will further improve the results. As it can be seen, the presented pipeline satisfies all requirements discussed in section 5.3, also it provides clear steps as well as general solution for image synthesis for surface inspection.

4.7 Conclusion

The lack of images containing defects is a common obstacle in industrial inspection. In this work, we have presented a pipeline for synthesizing images of an object with both ideal and defective surfaces. It consists of three modules. The first module, Errsmith, is responsible for imprinting the object geometry with surface defects of various sizes and positions. This approach ensures valid light response around defective areas of the surface, which seems impossible when augmenting defects from real images to defect-free images. The second module, Callistemon, performs realistic image synthesis which incorporates surface inspection data along with material, light,

and camera specification to ensure realistic appearance. The third module provides the acquisition of images through a robotic system and ensures that the geometry of the objects in synthesized images and real images match. This pipeline promises numerous applications. One example is synthesizing images containing features unseen in real-world data for machine learning approaches. It can be also used for simulating inspection systems while designing specialized hand-crafted algorithms. Also, it can help during viewpoint optimization for surface coverage where it is required to know if a certain defect is visible from a particular camera and light position and orientation. For further work, we would point out the importance of automatic evaluation of the synthesized image with regards to the real image and the metric which would allow this. If synthesizing images containing features unseen in real data would help defect detection models learn to detect surface defects, could inspection systems start developing in similar directions?

Core references

Lovro Bosnar, Doria Saric, Siddhartha Dutta, Thomas Weibel, Markus Rauhut, Hans Hagen and Petra Gospodnetic, "Image synthesis pipeline for surface inspection", LEVIA 2020: Leipzig Symposium on Visualization in Applications, Leipzig, Germany, *doi:* `10.31219/osf.io/kqt8w`

Chapter 5

Procedural Texture Synthesis for Surface Inspection

The automated visual surface inspection planning is an important part of the quality assurance in automated custom product manufacturing. Visual surface inspection planning tackles image acquisition design and defect detection. Both tasks greatly benefit from the utilization of realistic and automated image synthesis of the inspected object. The realism of synthesized images greatly depends on object material, whose properties are largely influenced by texture. In this work, the focus is on parametric texture synthesis and its application for visual surface inspection planning. First, the texture present on physical samples is analyzed and the requirements for texture synthesis models in visual surface inspection are introduced. Based on observation and surface characterization standards, a model capable of reproducing texture on physical samples is presented. This approach is generalized and further models are presented with respect to requirements. Finally, the importance of surface texture from the visual inspection planning perspective is highlighted.

The contributions of this work are (1) texture synthesis model requirements for surface inspection, (2) novel texture synthesis models which enable inspection planning expert to generate common machined surface texture in repeatable and consistent manner, (3) introducing surface metrology concepts in computer graphics texture modeling, (4) series of simulations with introduced texture synthesis models to highlight and discuss the importance of surface texture influence on object appearance and thus visual surface inspection planning.

5.1 Introduction

The goal of automated visual surface inspection is to detect various defects on a product's surface. This process requires a complete visual coverage of the inspected product surface and usage of robust image processing algorithms for defect detection. Industry 4.0 incentivizes product customization, which makes the development of inspection procedures very challenging. An answer to this likely lies in virtual inspection planning which is an actively researched area (Gospodnetic et al. [6], [188], [189] and Mosbach et al. [190]).

The virtual inspection planning for a visual inspection aims to solve two main tasks: the spatial configuration of acquisition hardware to ensure required product coverage and the development of robust image processing algorithms for defect detection. Both tasks rely heavily on the acquisition of the representative product image data, i.e., the images showing the appearance of the object during the inspection. As shown by Mohammadikaji [191], [192] and Jorgensen et al. [193] the use of photo-realistic rendering for a representative image simulation can greatly benefit the inspection design process.

In Chapter 4 a pipeline for image synthesis as a part of a virtual inspection planning system was presented. Using this pipeline, an inspection planning expert, describes a 3D scene with the particular spatial configuration of the inspected object, expected defects and acquisition hardware (illumination and camera) and synthesizes images using physically-based light transport simulation (Hall et al. [194]). Synthesized images are then used for assessing the the acquisition hardware configuration and its placement for obtaining the required coverage as well as development of defect detection algorithms. In this work, the focus is on the texture of the inspected surface, to which, the hardware positioning and the image processing algorithms are very sensitive due to visible surface patterns, highlights and shadows (Fig. 5.8). Therefore, texture synthesis models are presented which are integrated into the image synthesis system for surface inspection presented in Chapter 4, where inspection planning expert must only choose texture parameters. Presented texturing modes enable photo-realistic image synthesis and thus generating images as they would be taken in real-world, allowing inspection expert to fully virtually perform the inspection planning.

Texture synthesis methods developed for computer graphics so far are mostly targeting the film and the game industries and are undoubtedly successful in achieving a high degree of realism. However, the existing texture synthesis paradigm aims to assist artists in their creation process in which the textures are manually and subjectively created and applied onto objects. On the other hand, texture simulation for the visual inspection planning process must be objective, parameterized, automatic and must assist inspection planning expert. Therefore, the manual texture creation and application is out of the question as it is too time-consuming and not consistently repeat-

able.

To tackle the texture synthesis for the visual surface inspection planning, a machining surface pattern of physical gear and spring inspection objects (Fig. 5.5 and 5.6) is analyzed. In discussion with the domain expert, the texturing model requirements are gathered. Furthermore, a novel texture synthesis model capable of reproducing the pattern present in the gear and spring inspection objects, while satisfying the presented requirements is presented. In order to model a physically correct texture, surface metrology concepts were used that are, to our knowledge, used for the first time in computer graphics texture modeling. Those foundations are generalized to the parallel and the radial textures (Fig. 5.1 and 5.10) that are often present in machined surfaces. Finally, the importance of texture for visual surface inspection planning is highlighted by providing and discussing simulations with presented models.

The contributions of this work are: (1) the requirements for texture synthesis models for surface inspection (Section 5.3), (2) the integration of the surface metrology concepts in computer graphics texture modeling (Sections 5.2 and 5.4), (3) three novel procedural texture synthesis models (Section 5.4) that can be integrated into the rendering engines, providing parameters for the planning experts to perform an automated generation of realistic textures in a repeatable manner, without any knowledge of computer graphics, (4) simulations highlighting the importance of the realistic surface texture for virtual inspection planning (Sections 5.5 and 5.6).

5.2 Related Work and State of the Art

Image synthesis and material modeling

A synthesized image is said to be realistic if it cannot be *perceptually* distinguished from a real image or environment (Greenberg et al. [176]). On the other hand, a synthesized image is said to be *photo-correct* if it is indistinguishable from the real image based on *physical* foundations. The foundations for the realistic image synthesis are a physically correct 3D scene definition and a physically-based rendering (e.g., appleseed [184]). Physically rendering is widely researched and discussed by Dutre et al. [195]. In this work, we focus on the material modeling aspect of the 3D scene definition, namely texture modeling.

The foundation for physically plausible material modeling is the microfacet-based bidirectional reflectance distribution function (BRDF) discussed by Walter et al. [9]. Microfacet-based BRDF computes the amount of reflected light for a given view, light direction and surface properties. It models cumulative effect of eye-invisible surface geometry which greatly affects the appearance. Microfacet BRDF consists of three main parts: distribution of microfacet normals (NDF), geometry term and Fresnel term. NDF describes

amount of microfacet normals which contribute to the light reflection. It is represented as a continuous distribution parameterized by roughness value which determines microstructure regularity. Geometry term describes which microfacets are shadowing and masking incoming and reflected light as discussed by Heitz [38]. Finally, Fresnel term describes amount of reflected light and thus tint of the reflection.

Spatially uniform BRDF properties over surface result in an unrealistically smooth surface. The key for achieving surface complexity and therefore realism is to spatially variate BRDF properties (i.e. roughness and normals) over a 3D model surface using textures. Spatial variation of BRDF parameters over a the surface significantly contributes to the realistic appearance (see Fig. 5.10) and thus is the focus of our work.

Texture synthesis

For the purpose of the visual surface inspection planning, we are interested in both a realistic and an automated texture synthesis. Therefore, the existing texture synthesis methods can be grouped by the level of automation: manual, example-based and procedural. Since manual object texturing requires artists (e.g. Burley [52]), we consider such approaches unsuitable for an automated environment and will not consider them.

The example-based approaches, surveyed by Wei et al. [56], replicate the pattern specified by an exemplar image, over an object. However, a parameterized mesh surface is required but typically not readily available. Parameterization of complex meshes often requires a certain degree of manual work and therefore is unsuitable for an automated environment. Different approaches utilize generative adversarial networks (GAN) to synthesize a large-scale output image while maintaining both the small-scale and the global characteristics of the exemplar [196] and [197]. Unfortunately, applying resulting images over free-form geometry rises the problem of image stitching and repetition. Next, visual artifacts can often appear in the generated tiles or in the corners and the borders of the final image which can be only tackled with time-consuming, trial and error-based training using a large number of image samples are required to be performed by an experienced user. Moreover, the representative exemplars, from which the replicated features are learned, can be challenging to obtain.

The procedural approaches do not require the mesh parameterization and a texture pattern is algorithmically defined using the rendering engine shading languages, e.g. Open Shading Language (OSL) [185]. Procedural textures can cover arbitrarily large surfaces with patterns ranging from stationary to globally varying and are configured by a set of the predefined parameters representing the minimal user input (Ebert et al. [62]). Therefore, they allow a high degree of automation, making them well suited for the automated visual surface inspection planning. In our work, noise-based

textures (Lagae et al. [198]) proved useful for introducing complexity in the encoded pattern. Approach by Glanville [109] where texture elements are placed on the surface points determined by a procedural point distribution, inspired our method, but instead of a random placement of elements, we encode their positions in order to obtain a more structured pattern.

The development of procedural textures complexity intensifies with the texture and geometrical complexity. We make this problem more tractable by restricting the scope of textures to the well standardized textures resulting from the machining processes (Section 5.2).

Grooves and anisotropic reflectance modeling

Vangorp et al. [199] pointed out the influence of surface texture on light reflection and thus highlights. Manufactured surfaces, as discussed by Bosch [200] exhibit scratches or grooves that can be individually invisible, meaning that they are observed as an average anisotropic reflectance or individually observable as detailed shadows and highlights. Directionality and orientation of grooves determine the anisotropic reflectance as shown by Poulin et al. [201].

Raymond et al. [202] simulated the average anisotropic reflectance by stretching the BRDF lobe in a desired tangential direction. Poulin and Fournier [201] simulate the average anisotropic reflection using eye-invisible cylinders representing surface grooves. Unfortunately, the lack of visible geometrical features makes the surface appearance unrealistically smooth. Nevertheless, this approach inspired us to use cylinders and torus as primitives for modeling grooves. Data based approach was performed by Dong et al. [174] where a microfacet BRDF was constructed using the surface profilometer measurements. The resulting model reproduced the average anisotropic reflectance and the visible texture details were additionally added via a Gaussian random field. In contrast, our approach does not require a measurement and directly results in an anisotropic reflectance with visible texture details.

Bosch [203–206] modeled the individually visible scratches by utilizing two BRDFs and a 2D texture for marking the scratch positions. This approach requires a 2D scratch texture creation. Raymond et al. [207] modeled a scratched surface by stacking layers with different scratch distributions. Each layer (i.e. 2D texture) consists of an array of individual scratches with pre-computed reflections. Unfortunately, the resulting surface is not a result of any specific manufacturing process. Both approaches require a UV parametrization of the mesh for texture mapping.

Manufacturing and surface texture analysis

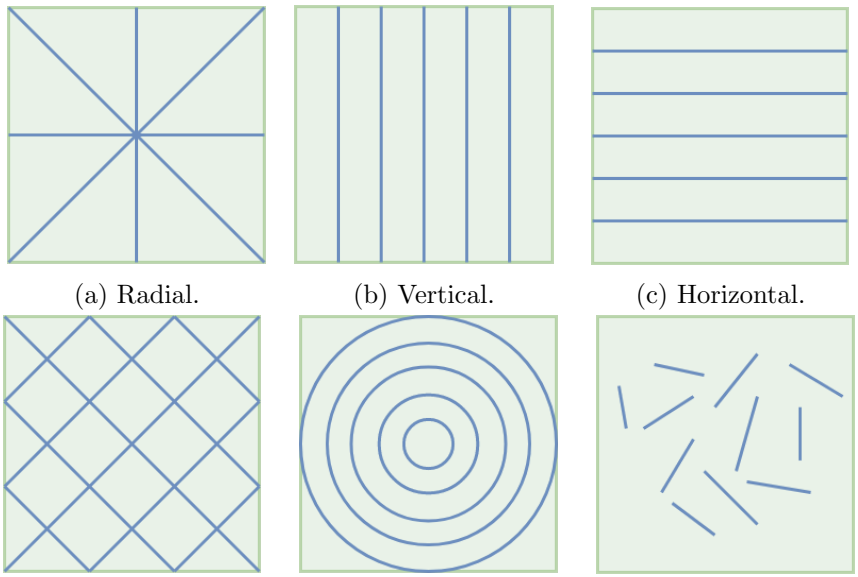
The surface texture greatly depends on the manufacturing process. A good overview of manufacturing processes is given by Black et al. [2] where they are grouped into casting, forming and machining processes. The casting processes result in a bumpy surface similar to the skin of an orange. The machining processes such as turning, milling and abrasion result in an elongated grooved surface texture defined by the cutting tool [208]. In a turning process, a workpiece is rotated against a cutting tool. In a milling process, a rotating cutting tool is used for surface shaping. The abrasive methods shape a surface using small abrasive granules. Additionally, the surface can be processed further in order to achieve certain aesthetic or functional properties.

Black et al. [2] and Jiang et al. [209, 210] further give a good overview of the surface metrology frameworks and the surface characterization, namely roughness, waviness and lay (i.e., form). Surface roughness refers to the finely spaced surface irregularities. Waviness describes the surface irregularities with spacing greater than that of surface roughness. Lay is determined by the production method and it describes a predominant surface pattern direction. Lay types (Fig. 5.1) standardised by ISO 1302 [211] inspired the texture synthesis models in our work. Wolf [212] provides an overview of surface representation models as well as surface characterization via field and feature areal parameters used in surface metrology for scale-limited surfaces.

Surface texture analysis is a widely researched aspect of machine vision (see Beyerer et al. [5]). The texture of the manufactured surface can be viewed as structural or semi-structural. A structural texture contains the elementary patterns aligned in a deterministic spatial pattern. On the other hand, the elementary patterns in semi-structured texture can be still recognized but their placement is affected by stochastic variations. For the texture modeling, we were inspired by viewing texture as a set of elements and their placement. The structural texture description (Humeau [213]) perceives textures in terms of elements and their placements. Another approach that inspired our work is Germ-grain model developed in continuum percolation theory. Germ-grain model is a generalization of Boolean and Boolean-Poisson models. It is driven by arbitrary stationary point process which assigns arbitrary compact sets to points [108].

5.3 Requirements

The existing computer graphics methods provide an excellent foundation for realistic image synthesis. The realism is achieved partly by the physically-based light transport simulation and surface reflection models. The other part of realism is achieved through the work of artists. Without the involvement of an artist who creates the fine-scale surface details, materials and



(a) Radial. (b) Vertical. (c) Horizontal.

(d) Crossed. (e) Circular. (f) Isotropic.

Figure 5.1: Surface lay types (see Black et al. [2]).

imperfections, surfaces are unnaturally smooth and too perfect, thereby not serving as a good representation of real surfaces. Visual surface inspection planning requires a consistent and repeatable automated image synthesis. Therefore, an automated and parameterized texture synthesis is needed. For this purpose the procedural texturing methods are most suitable because they enable a parameterized development of the texture synthesis models with a minimal and more intuitive user input. Based on those observations, a procedural texturing model must:

R4 Generate physically based texture feature values.

R5 Ensure a high resolution for different viewing distances.

R6 Cover an arbitrary surface area.

R7 Be configurable by a set of predetermined parameters.

R8 Require only intuitive input parameters.

R9 Be compatible with rendering engines.

The use of the physically-based surface values (**R4**) in a physically-based light transport and shading promise a physically plausible surface-light interaction which is, in turn, a prerequisite for a realistic appearance.

During a visual inspection, viewing distance can vary (e.g. variation of camera's focal length and its distance from the surface). Therefore, the texture synthesis model must support a high-resolution texture during rendering, regardless of viewing distance (**R5**).

Inspected products vary greatly in geometrical complexity. Therefore, texture synthesis methods must cover an arbitrarily large, complex and free-form surface area (**R6**).

The focus is on the machined surface textures which can be standardized and categorized into classes (e.g., circular brushing) as discussed in Black et al. [2]. Each class contains a range of textures that are similar yet distinguishable. For example, a two textures from the circular class may have a different groove size which makes them a two different instances of the same texture class. Therefore, the texture synthesis model must provide the configuration parameters for recreating a range of texture instances of a certain class (**R7**).

The texture synthesis models must be usable by quality inspection expert who is unlikely to be a computer graphics expert. Therefore, intuitive input parameters are needed (**R8**).

Texture synthesis models must provide a standardized interface to rendering engines used for image synthesis in virtual surface inspection planning (**R9**).

5.4 Methods

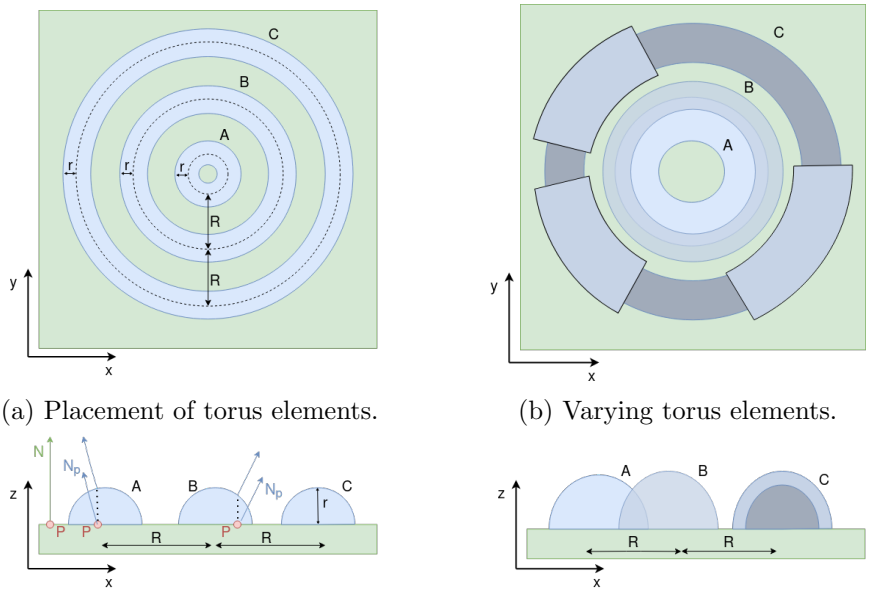
In this section, procedural models which describe visible texture patterns algorithmically are presented, enabling continuous texture over object surface evading problems of image stitching and tiling that have to be tackled in example-based and GAN approaches.

Firstly, three novel texture synthesis models for describing common machining surfaces, namely circular, radial and parallel are presented. Secondly, three models of which two are based on existing methods, namely Worley cellular texture for bumpy pattern and checker texture are shortly discussed. The third model is built via triangular waves to represent knurling pattern.

Circular texture synthesis model

A circular texture (Fig. 5.1e) is modeled using a large amount of torus elements placed concentrically around the world origin as illustrated in Fig. 5.2a. Each torus is defined by a minor (tube) radius r and a distance from the neighboring torus element R. If a shading point P is lying inside a torus tube radius, then a perturbed surface normal N_p in shading point P is calculated, otherwise, a mesh surface normal N is used during the shading (Fig. 5.2c). The process of evaluating the surface normal of a shading point

is given in Algorithm 2. The perturbed normal N_p in a shading point P is calculated using a tangent vector T: $N_p \leftarrow \alpha * T + (1 - \alpha) * N$. The tangent vector T is found using the closest torus element to the shading point P. The closest torus element with radius R_{major} is found by using the shading point location P_{xy0} and the regular placement of torus elements shown in Fig. 5.2a. The perturbed normal is interpolated between the original mesh normal N and the tangent T using a parameter $\alpha \in [0, 1]$. The interpolation parameter α dictates an amount of normal perturbation. We propose a smaller interpolation parameter (e.g., 0.3) so that the perturbed normal N_p does not significantly deviate from the mesh normal N. Ideal and identical torus elements are not a good representation of grooves resulting from machining because cutting tools always introduce deviation from the ideal grooving pattern. The same also holds for the cylinder elements in the parallel and radial texture synthesis methods. Therefore, additional pattern complexity is introduced by adding noise to the minor radius. The minor radius can vary per torus elements (see torus A and B in Fig. 5.2b and 5.2d) which is achieved by using torus major radius for noise evaluation. The minor radius can also vary per segments of a torus object (see torus C in Fig. 5.2b and 5.2d). In this case, the angular position of the torus element is used for noise evaluation. In this way, an overlapping of the torus elements is achieved, forming a more complex surface.



(a) Placement of torus elements.

(b) Varying torus elements.

(c) Normal vector evaluation.

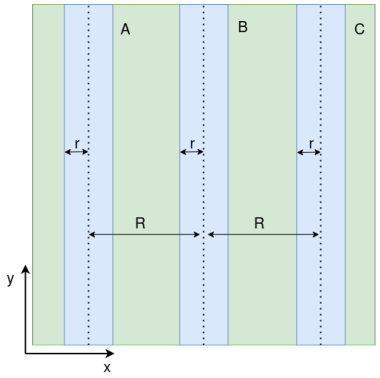
(d) Varying torus elements.

Figure 5.2: Circular texture model.

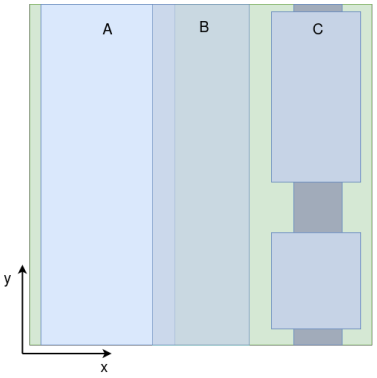
Algorithm 2 Circular texture synthesis model

Require: P, N, R, r
Ensure: N_p
  $P_{xy0} \leftarrow (P_x, P_y, 0)$
  $R_{major} \leftarrow R * \lfloor ||P_{xy0}||/R \rfloor$ ▷ current torus
  $R_{minor} \leftarrow r + noise()$ ▷ variation
  if $|||P_{xy0}|| - R_{major}| < R_{minor}$ **then**
    $\alpha \leftarrow |||P_{xy0}|| - R_{major}|/R_{minor}$
    $T \leftarrow P_{xy0} - norm(P_{xy0}) * R_{major}$ ▷ tangent
    $T \leftarrow norm(T)$
    $N_p \leftarrow \alpha * T + (1 - \alpha) * N$
    $N_p \leftarrow norm(N_p)$
  else
    $N_p \leftarrow N$

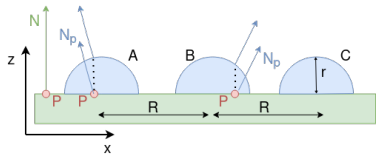
Parallel texture synthesis model



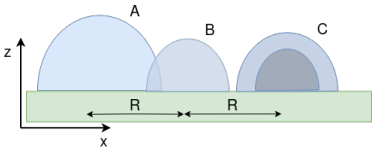
(a) Placement of cylinder elements.



(b) Varying cylinder elements.



(c) Normal vector evaluation.



(d) Varying cylinder elements.

Figure 5.3: Parallel texture model.

The parallel texture (Fig. 5.1b and 5.1c) is modeled using a large number of parallelly placed cylinder elements as illustrated in Fig. 5.3a. Each cylinder is defined by its radius r and its distance to the neighboring cylinders R. If the shading point P is inside the cylinder element, then a perturbed normal N_p is calculated and used for shading calculation. Otherwise, a mesh

normal N is used for the shading calculation (Fig. 5.3c). If desired, cylinders can be rotated around an axis parallel to the texture plane normal for a given angle θ. The evaluation of the surface normal in a shading point is described in Algorithm 3. The perturbed normal N_p is calculated using a tangent vector T as described for circular texture synthesis model. In this case, the closest cylinder with position $D_{cylinder}$ to the shading point P is found using the shading point position P_x and the regular placement of cylinder elements shown in Fig. 5.3a. The pattern complexity is increased by adding noise to the cylinder radius. One possibility is to vary radius per cylinder, that is, distance from shading point P to the cylinder center is used for noise evaluation (see cylinder elements A and B in Fig. 5.3b and 5.3d). The radius can also be varied per cylinder segments (see cylinder element C in Fig. 5.3b and 5.3d). In this case, the noise is evaluated using the shading point position in the cylinder. As a result of radius variation, the cylinder elements are likely to overlap thus creating a more complex surface texture.

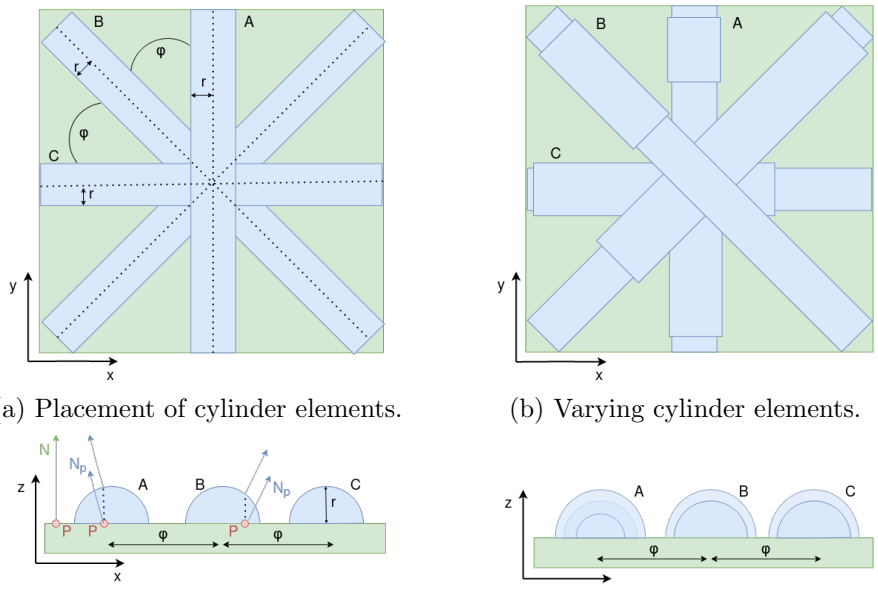
Algorithm 3 Parallel texture synthesis model

Require: P, N, R, r, θ
Ensure: N_p
  $P \leftarrow rotate(P, \theta, (0, 0, 1))$ ▷ rotate around z axis
  $D_{cylinder} \leftarrow R * \lfloor P_x/R \rfloor$ ▷ closest cylinder distance
  $R_{cylinder} \leftarrow r + noise()$ ▷ variation
  if $|P_x - D_{cylinder}| < R_{cylinder}$ **then**
    $\alpha \leftarrow |P_x - D_{cylinder}|/R_{cylinder}$
    $T \leftarrow (P_x - D_{cylinder}, 0, 0)$ ▷ tangent
    $T \leftarrow norm(T)$
    $N_p \leftarrow \alpha * T + (1 - \alpha) * N$
    $N_p \leftarrow norm(N_p)$
  else
    $N_p \leftarrow N$

Radial texture synthesis model

The radial texture (Fig. 5.1a) is modeled using a large amount of cylinder elements which are placed radially, as illustrated in Fig. 5.4a. Each cylinder is defined by its radius r and an angle to its neighboring cylinder ϕ. If the shading point P is inside the cylinder radius then a perturbed normal is calculated, otherwise, a mesh normal is used for the shading (Fig. 5.4c), as described in Algorithm 4. The perturbed normal N_p is calculated using vector T as described for circular texture synthesis model. The closest cylinder to P is constructed by finding a shading point angle from a reference direction V_{ref}. Pattern complexity is increased by adding noise to the cylinder radius. Noise is evaluated using the shading point position on the cylinder,

resulting in radius variation per cylinder segments (see Fig. 5.4b, 5.4d).



(a) Placement of cylinder elements.

(b) Varying cylinder elements.

(c) Normal vector evaluation.

(d) Varying cylinder elements.

Figure 5.4: Radial texture model.

Algorithm 4 Radial texture synthesis model

Require: P, N, ϕ, r
Ensure: N_p
  $V_{ref} \leftarrow (1, 0, 0)$ ▷ reference direction
  $P_{xy0} \leftarrow (P_x, P_y, 0)$
  $\phi_{P_{xy0}} \leftarrow \arccos(norm(P_{xy0}) \cdot V_{ref})$
  $\phi_{cylinder} \leftarrow \phi * \lfloor \phi_{P_{xy0}} / \phi \rfloor$ ▷ closest cylinder angle
  $V_{cylinder} \leftarrow (\cos(\phi_{cylinder}), \sin(\phi_{cylinder}), 0)$ ▷ direction
  $P_{cylinder} \leftarrow V_{cylinder} * (V_{cylinder} \cdot P_{xy0})$ ▷ projection
  $R_{cylinder} \leftarrow r + noise()$ ▷ variation
  if $||P_{cylinder} - P_{xy0}|| < R_{cylinder}$ **then**
    $\alpha \leftarrow ||P_{cylinder} - P_{xy0}|| / R_{cylinder}$
    $T \leftarrow P_{xy0} - P_{cylinder}$ ▷ tangent
    $T \leftarrow norm(T)$
    $N_p \leftarrow \alpha * T + (1 - \alpha) * N$
    $N_p \leftarrow norm(N_p)$
  else
    $N_p \leftarrow N$

Common and readily available models

Checker texture synthesis model Common procedural texture often available in modeling tools. This method is dividing object into regular cubes effectively resulting in checkerboard texture. Neighboring checkers differ in height values and roughness. Height value is used to generate perturbed normal in shading point using Mikkelsen's surface gradient-based bump mapping framework [65].

Bumpy texture synthesis model Procedural texture based on cellular basis function introduced by Worley [62] where the closest distance from the feature is used as height value. Height value is used for generating perturbed normals [65].

Knurling texture synthesis model This procedural texture is based on two triangle waves that are placed at different angles. The maximum value of triangle waves in every point is used as a height value. The height value is used for generating perturbed normals [65].

5.5 Results

Models introduced in Section 5.4 were used in a simulated 3D surface inspection scene containing inspected object, illumination and camera (based on the description given in Chapter 4). The 3D scene is rendered using appleseed rendering engine which is an offline, physically-based, production rendering engine. For rendering, path tracing light transport with 150 samples per pixel was used.

Inspected objects were represented by gear, spring and hirth geometry (Fig. 5.6). In addition, the gear object contains mesh imprinted scratch and dent defects which are examples of defects commonly required to be detected. All mesh objects have uniformly sized and distributed triangles across the surface. Gear, hirth and spring meshes are composed of 113K, 44K and 1M triangles respectively. Gear and hirth objects are CAD models created to resemble real inspected parts, whereas the spring object is created to resemble a challenging, free-form surface. For both gear and the spring object, a physical sample was created based on CAD description and was used to obtain real images (see Fig. 5.5). The acquisition is based on the description given in Chapter 4 and performed using Prosilica GC 2450 camera with a Sony ICX625 CCD sensor (2448×2050 pixels) - an industrial, grayscale camera with mounted Kowa 12mm lens. Material is defined by appleseed's metal BRDF model and texture synthesis models introduced in Section 5.4. Texture synthesis models are implemented in OSL, which also provides a set of predefined noise functions. *cellnoise*() was used to achieve torus and cylinder radius variation (see Section 5.4). As described in OSL [185], *cellnoise*() is a discrete function constant on $[i, i + 1)$ for all integers i with different and uncorrelated value at every integer. The origin

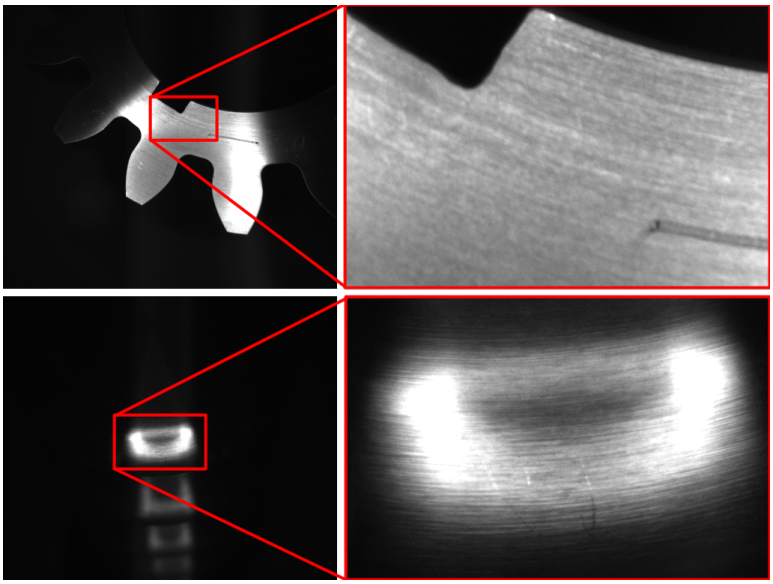


Figure 5.5: Real images of spring (top) and gear (bottom) objects with zoom on surface pattern.

of the simulated objects corresponds with the world origin of the 3D scene, therefore circular and radial textures are centered (i.e., seeded) in world origin. Generally, center of circular and radial texture depends on object shape. Therefore, it should be set manually. An automatic solution would be using the center of the inspected object bounding volume.

The illumination source is defined by torus-shaped mesh geometry and diffuse-emission material and placed in a black, non-emissive environment. The emissive material is specified using OSL. Light position is relative to the camera, meaning that it is mounted on the camera and its position transformation follows the camera transformation.

The camera is defined by appleseed's pinhole camera model with a pixel

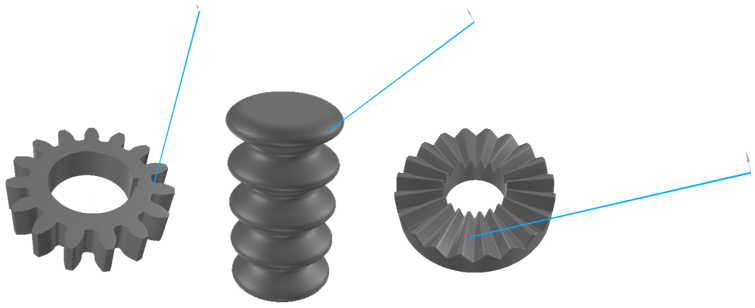


Figure 5.6: Mesh objects (gear, spring and hirth) and corresponding view directions.

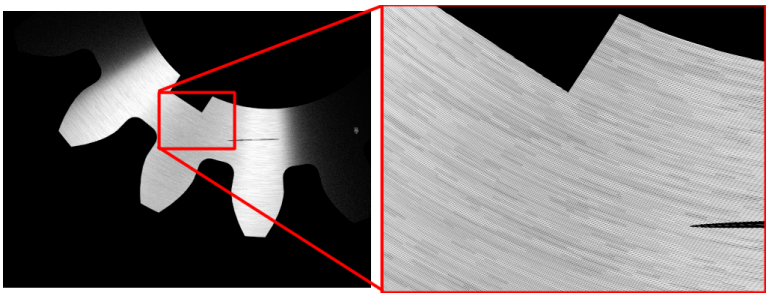


Figure 5.7: Gear object with circular texture and increased camera focal length (i.e. zoom on surface pattern).

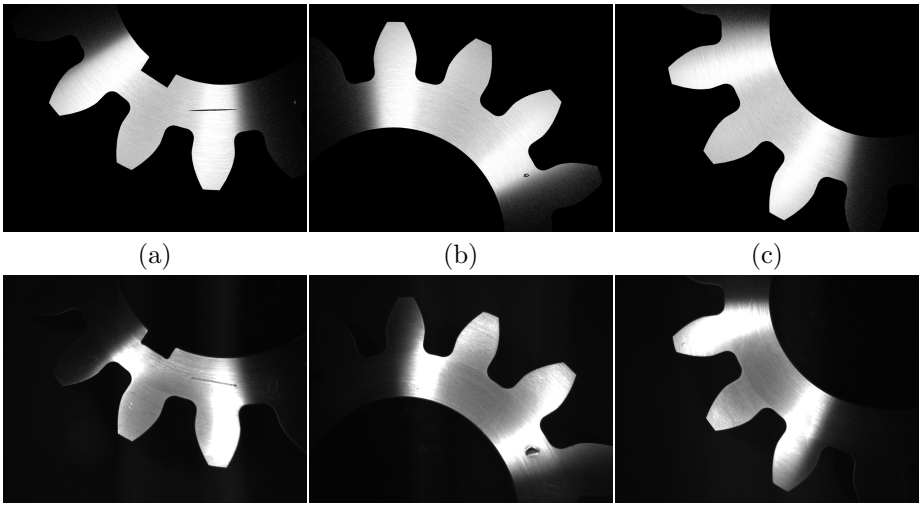


Figure 5.8: Comparison of synthesized images (top) and real acquired images (bottom) of gear objects with circular texture.

size of 0.00824 mm, focal length 12.93 mm and resolution 1024x768 pixels. The parameters correlate with real camera parameters.

Comparison of synthesized and real images is given in Fig. 5.8 and 5.9. The simulations in Fig. 5.10 contain set of images for each model, with viewing direction depicted in Fig. 5.6. All scene parameters are fixed, except for the texture showing the importance of texture influence on object appearance. Circular, parallel and radial textures are presented in Fig. 5.10a to 5.10c. Circular textures with increasing torus minor radius and a distance between torus elements are presented in Fig. 5.10d to 5.10f. Fig. 5.7 shows circular texture rendered with focal of 60 mm, effectively zooming in on the surface. Finally, Fig. 5.11 shows influence of different texture on defect visibility.

5.6 Discussion

The presented texture synthesis models for circular, parallel and radial textures will be discussed. Also, a series of simulations where all scene parameters are fixed except for the texture (Fig. 5.12, 5.10) will be presented. The goal of these simulations is to highlight the importance of texture on appearance and visual surface inspection planning.

5.6.1 Texture Synthesis Models

The presented methods define geometrical surface properties (i.e., normal vectors) based on parameters resulting from machining processes. Therefore, instead of subjective texture generated manually by an artist, surface texture in rendered objects is a result of parameterized calculations founded on surface manufacturing standards, partially satisfying requirement **R4**. Therefore, further research of physically correct surface texture parameters is needed.

In Fig. 5.7 is visible that a higher camera zoom level (i.e., higher camera focal length) produces high quality surface features resolution which satisfies requirement **R5**. Furthermore, Fig. 5.10 show that procedural texture can cover arbitrary surface area without repetition and stitching artifacts, also without problems caused by free-form surface. Therefore, requirement **R6** is satisfied. Both of the aforementioned observations are possible because the texture pattern is described algorithmically and is re-evaluated for different viewing conditions, in contrast to example-based or deep learning (e.g., GAN) approaches where only a static image of fixed size is generated.

Circular, parallel and radial (Fig. 5.10a to 5.10c) texture synthesis models represent three common machined surface texture classes (Fig. 5.1). For circular texture class, different texture instances are depicted (Fig. 5.10d to 5.10f) showing that the presented models are configurable by a set of predetermined parameters, satisfying requirement **R7**. Therefore, inspection planning experts have direct control over generated texture whereas in example-based or deep learning approaches, the user can only rely on image samples of which representative ones can be hard to obtain and time-consuming, experience-based training of a black box.

The presented texture models use only shading point position and normal known during rendering. No mesh preprocessing or any other user input is needed. Models provide parameters that are configuring texture elements and their placement, for which computer graphics knowledge is not needed, making models more suitable for inspection planning expert (requirement **R8**). Comparing models presented in this work to example-based and GAN approaches discussed in Section 5.2, it can be seen that the texture can be generated with considerably less user input because dataset acquisition, preparation and training is not required to be performed by the inspection

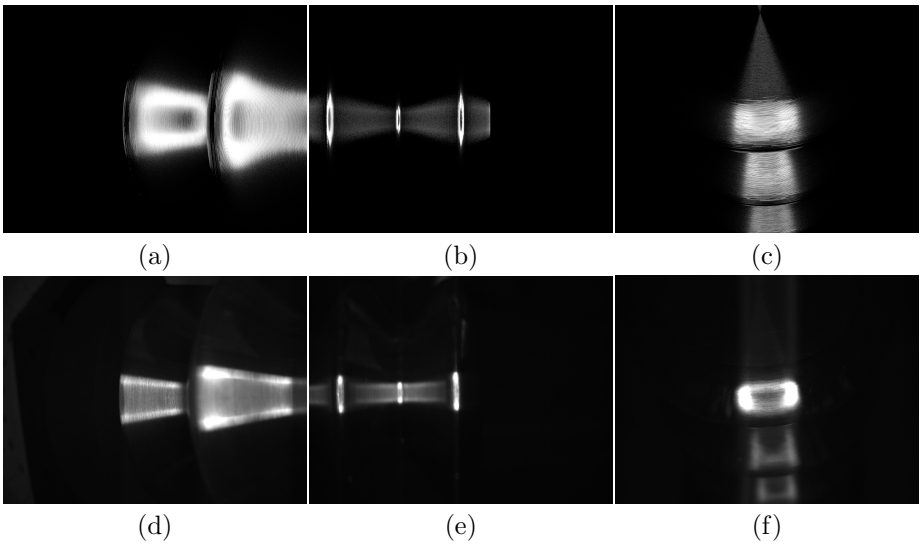
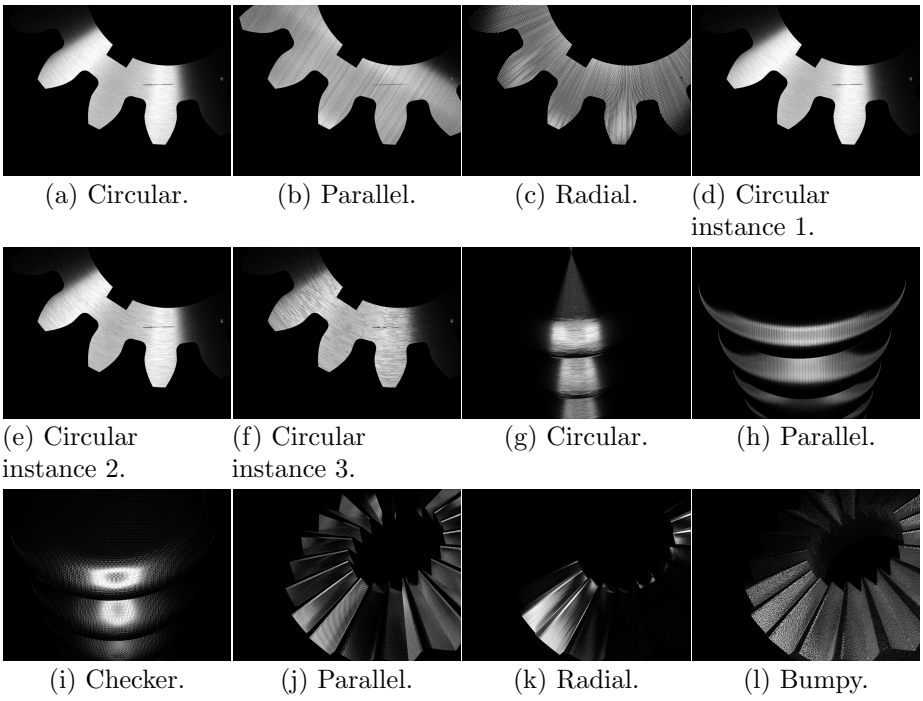


Figure 5.9: Comparison of synthesized images (top) and real acquired images (bottom) of spring objects with circular texture.

planning expert. Furthermore, texture evaluation time is not affected by the number of texture elements (e.g., torus and cylinder elements) that contribute to texture detail and it is typically under one second for a personal laptop (CPU: Intel Core i7, 8 cores, 1.9GHz. GPU: Intel. RAM: 16GB). This is because only relevant texture element is constructed for current shading points (see Section 5.4).

Models are implemented using OSL, a highly optimized, industry-standard shading language supported by physically-based, ray-tracing rendering engines. This satisfies the requirement **R9**. As OSL syntax is very similar to other shading languages, implementation transfer to other rendering environments is easier, should there be a need for it.

The parameters of the circular texture synthesis model can be fitted so that anisotropic reflectance and texture patterns of the synthesized object matches the real object to a high degree resulting in a close resemblance (see Fig. 5.8 and 5.9). Based on the same principles as in the circular texture synthesis method and with the help of manufacturing standardizations (Fig. 5.1), parallel and radial texture synthesis models are developed as discussed in Section 5.4. Therefore, inspection planning expert can fit the parameters of given models to match the appearance of the real texture. Differences introduced by factors such as surface defects (e.g., scratches), illumination, camera lens and sensor can still be observed (e.g., reflection artifact in Fig. 5.9f) and should be tackled in further work, but are not within the scope of this work. Also, it is important to note that background can be barely visible in acquired images, which is not the case in inspection acquisition sys-



(a) Circular. (b) Parallel. (c) Radial. (d) Circular instance 1.

(e) Circular instance 2. (f) Circular instance 3. (g) Circular. (h) Parallel.

(i) Checker. (j) Parallel. (k) Radial. (l) Bumpy.

Figure 5.10: Gear, spring and hirth objects. Note that only a difference in texture causes a huge difference in surface visibility.

tems, thus can be ignored. Additionally, further investigation of parameters resulting from manufacturing processes (e.g., groove amplitudes) as well as parametric relation between machining and texture synthesis on arbitrary surfaces is needed.

5.6.2 Texture Influence on Appearance

Uniform, smooth surfaces without textures will result in simple, isotropic reflection. On the other hand, surface patterns such as grooves cause different types of anisotropic reflections depending on groove orientations and shapes as discussed in Section 5.2. This effect is visible in Fig. 5.9 and 5.10. It is important to note that anisotropic reflection in presented figures is purely due to visible geometrical features, generated by presented texture synthesis models as opposed to average anisotropy reflectance models, discussed in Section 5.2. Differences in reflection due to surface texture must be taken into account during surface inspection because some parts of the surface will not be visible. Furthermore, visible geometrical details are the key to a realistic surface and have additional importance when it comes to visual inspection planning, which is a possibility to adapt the acquisition conditions without ever physically positioning the hardware. This way, an inspection expert can truly plan what will the acquisition capture.

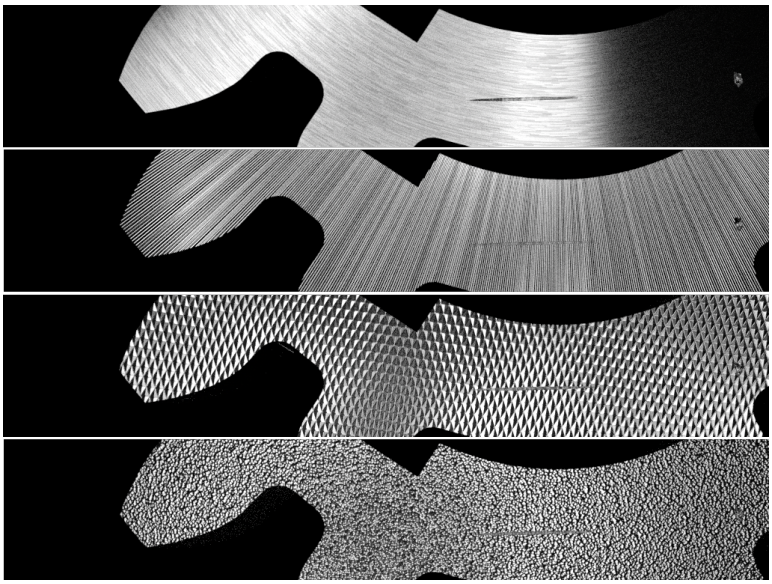


Figure 5.11: Gear object with circular, radial, knurling and casting textures illustrating how defect visibility greatly depends on texture.

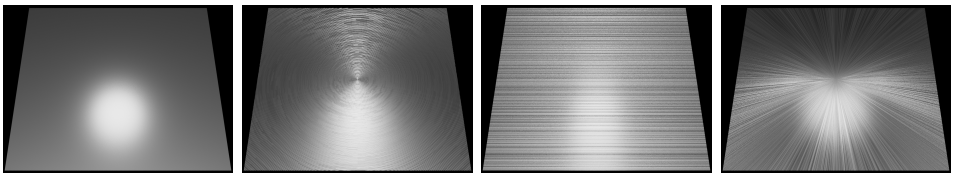


Figure 5.12: 3D scene containing the same setup except for surface textures: no texture (only BRDF), circular texture, parallel texture, radial texture. Note how various surface textures cause various anisotropic reflections.

Surface texture causes a wide range of highlights and shadows as it can be seen in Fig. 5.10. Due to the highlights and shadows, the visibility of surface parts changes gradually, therefore some parts of the surface are visible, while others are hidden. This leads to the problem of defect visibility (see Fig. 5.11) where scratch and dent defects are ranging from visible to invisible based on the texture. Finally, presented simulations clearly show how the difference in texture causes a wide range of appearance levels.

5.7 Conclusion

In this work, we have tackled texture synthesis used in representative product image data generation for visual surface inspection. We have introduced requirements as well as presented three novel procedural texture synthesis models capable of generating texture patterns common in machined sur-

faces. Presented models allow surface inspection planning expert to fully virtually visualize object surface for inspection planning without computer graphics knowledge. Presented models are based on procedural recreation of surface geometrical pattern, on a greater scale than BRDF, for describing realistic continuous surface texture. Furthermore, presented models ensure introduced requirements, namely, they generate physically based values **R4**, ensure high resolution regardless of view distance **R5**, cover arbitrary large surface area **R6**, enable multiple texture instances per texture class **R7**, require only input intuitive to non-experts **R8** and are compatible with a range of existing rendering engines **R9**. With presented models, inspection planning experts can create realistic machined surfaces without knowledge of computer graphics, in a repeatable and automated manner. Finally, the importance of surface texture for visual surface inspection planning has been highlighted by providing and discussing series of simulations with the presented texture synthesis models. In future work, dynamic texture scale evaluation should be investigated.

Core references

Lovro Bosnar, Markus Rauhut, Hans Hagen and Petra Gospodnetic, "Texture synthesis for surface inspection", LEVIA 2022: Leipzig Symposium on Visualization in Applications, Leipzig, Germany, *doi:* `10.36730/2022.1.levia.4`

Chapter 6

Procedural Defect Modeling for Virtual Surface Inspection

Development of automated visual surface inspection systems heavily depends on the availability of defected product samples. Both inspection hardware configuration and training of defect detection models require diversified, representative and precisely annotated data. Reliable training data of sufficient size is frequently challenging to obtain. Using virtual environments, it is possible to simulate defected products which would serve both for configuration of acquisition hardware as well as for generation of required datasets. In this work, parameterized models for adaptable simulation of geometrical defects, based on procedural methods are presented. Presented models are suitable for creating defected products in virtual surface inspection planning environments. As such, they enable inspection planning experts to assess defect visibility for various configurations of acquisition hardware. Finally, the presented method enables pixel-precise annotations alongside image synthesis for the creation of training-ready datasets.

Contribution of this work is presented in the video on the following link: `https://www.computer.org/csdl/video-library/video/1MgrjWE4J7W`

6.1 Introduction

Advancements in production are aiming toward the implementation of Industry 4.0 concepts such as automated and customized product manufacturing. However, the automation of production is limited by quality inspection and its level of automation. An important and widely used quality assurance method for ensuring defect-free product surfaces is visual surface inspection (VSI).

Automated VSI requires an inspection planning expert to configure acquisition hardware (i.e., camera and illumination) and its placement for required surface coverage. Due to the geometrical complexity of products and highly reflective materials, the inspection planning procedure is cumbersome and expensive. A solution was proposed by Gospodnetic et al. [6] in the form of the virtual inspection planning environment, where an expert configures the acquisition hardware and its placement in a 3D scene. The extension was introduced in Chapter 4, where the inspection environment was simulated using physically based rendering enabling representative image data synthesis and showing what a real camera in a real environment would see. This provided the expert with the possibility to predict acquired image characteristics without the need for a physical laboratory.

The discussed virtual inspection environment proposed an initial method to defect simulation. However, without the possibility of straightforward defect parameter adaptation which is crucial for the development of automated and robust VSI systems. First, various defect shapes exhibit different responses to illumination and thus visibility. Defect visibility is important information for inspection planning expert during the configuration of acquisition hardware placement (i.e., required imaging angle). Furthermore, VSI systems require machine vision models for defect detection. In the case of classical approaches, a redundant set of representative defects is needed for the development of robust analysis methods. On the other hand, machine learning approaches require large, diversified and precisely annotated datasets. Due to the lack of defected physical product samples, particular defects (which also have to be acquired) and the influence of a complex environment obtaining the right amount of images from the right position becomes difficult very fast. Furthermore, annotating the acquired images requires manual effort which is expensive, error-prone, inconsistent and often subjective.

In this work, the method introduced in Chapter 4 is extended with parameterized models for the adaptable generation of the geometrical dent and scratch defects on a simulated inspected object surface. Dent and scratch defects represent geometrical flaws on object which must be detected using visual surface inspection (Black et al. [2], ch. 43). Models for creating dent and scratch defects are obtained by using procedural geometry methods in order to completely algorithmically describe defect shapes, their distribution as well as integration of domain knowledge (e.g., manufacturing). To our knowledge, such complete control over defect modeling enabling integration of domain knowledge has been done for the first time. In this work, the focus is on manufactured metal parts of complex geometry and non-trivial texture, for which dent and scratch defects must be detected. Specifically, defecting workflow was demonstrated on metal car part - clutch (Fig. 6.6, 6.8 and 6.9).

Presented models enable controllable creation of defected product ob-

jects. Defects are geometrically imprinted into the object which ensures correct defect visibility under arbitrary view and illumination conditions. This provides representative defect visibility for an inspection planning expert during acquisition configuration. Finally, the presented method makes it possible to guarantee pixel-precise defect annotations simultaneously with image synthesis, thus enabling the automated creation of training-ready datasets.

This work introduces: (1) requirements for parameterized defect generation models, (2) procedural geometry modeling workflow for developing parameterized defecting models, (3) parameterized models for dent and scratch defects creation over object surface intuitive to inspection planning expert, (4) automated generation of defect annotations for training-ready datasets, (5) examples and discussion highlighting the importance of geometrical defect modeling for correct defect visibility representation.

6.2 Related Work and State of the Art

Generation of defected datasets can be performed using virtual environments or image-based methods. Virtual environments proved beneficial for both the generation of training datasets (Nikolenko et al. [214]) and the VSI planning process (Gospodnetic et al. [188]). Defect detection in VSI is based on illuminating the surface from different angles and observing its response. Thus, the visibility of surface flaws greatly depends on their geometrical nature (Black et al. [2], Chapter 43). Therefore, the focus is on modeling geometrical defects for virtual environments. For the sake of completeness, texture-based defects and their shortcomings in virtual environments are discussed. Finally, image-based methods and their drawbacks compared to virtual environments are discussed.

Geometry-based defects

Geometry modeling is widely adopted and greatly developed for building virtual environments for games and films. Powerful procedural modeling tools such as Houdini [107] are developed for flexibility and artist productivity. However, due to their complexity and learning curve, they are not suitable for inspection planning experts. Nevertheless, Houdini's procedural workflow inspired the approach presented in this work. In this work parameterized models for defect creation which are more suitable for usage in virtual inspection environments are presented.

Gutierrez et al. [215] used height maps for creating surface displacement representing a defect. However, the usage of height maps requires the unwrapping of geometry which requires substantial manual work for free-form geometry. Further, as displacement is done in the surface normal direction, it can not reproduce overhanging geometry.

In Chapter 4 a novel defect modeling workflow is presented, where a 3D geometry representing a defecting tool is modeled and imprinted on product surface geometry. Defecting tool geometry is imprinted into product surface using Boolean operations discussed by Zhou et al. [216]. Geometrically imprinted defects realistically and correctly represent 3D defect shape and its interaction with light. Geometrical defects can trap and reflect light in various directions based on viewing and illumination conditions. These effects can not be correctly simulated using texture-based or image-based defects as discussed by Rushmeier et al. [217]. In Schmedemann et al. [218], a similar workflow was used, however, it is not clear how were the defects placed or modeled and if any parameters are available. It can be only concluded that the defecting tool creation is mostly manual and limited to spherical shapes. The method introduced in Chapter 4 is extended by introducing parameterized and adaptable models for both the creation of defecting tools (i.e., dents and scratches) and their placement over the object surface.

Our approach is based on procedural methods. Foundations for procedural modeling are discussed by Ebert et al. [62]. The basis of procedural modeling lies in controllable stochastic methods based on noise (Lagae et al. [198]) which were utilized for defecting tool creation and placement.

Procedural geometry modeling is often used for content creation in virtual worlds (Smelik et al. [96]). Our work was specifically inspired by modeling terrain irregularities (Weiss et al. [111]), where modeling geometrical operations (e.g., subdivision and extrusion) proved as a good foundation for modeling the irregularities of the defecting tool geometry.

Texture-based defects

Modeling defects using procedural textures was discussed by Boikov et al. [219]. Defects were modeled by a series of deformations imposed on noise function. A similar approach, but with image textures was done by Bosch et al. [220]. The manually created texture was used to specify the position of scratches over the surface. This texture is then used to switch between two BRDFs: one for scratches and one for the surface. Different approach was taken by Raymond et al. [207] who modeled a scratched surface by stacking layers with different scratch distributions. Although impressive, each layer containing individual scratches reflections must be pre-computed and stored as 2D texture which further has to be applied on free-form surface. Unfortunately, manual involvement and image mapping on free-form geometry are not suitable for inspection planning expert. Furthermore, texture-based methods are not perturbing the actual geometry. Therefore, they fail to correctly model the defect geometrical details and depth. As interaction with light greatly depends on geometrical shape (e.g., light trapping or redirection) texture-based methods do not provide representative surface visibility

(Rushmeier et al. [217]) and thus defect visibility for all illumination and viewing conditions. In our work, defects are geometrically modeled ensuring correct interactions with light.

Image-based defects

Image processing methods have also been used for defect creation. Most successful approaches are based on GANs as discussed by Niu et al. [221]. However, there are several downsides. First, these approaches operate directly on images that must be acquired beforehand. Next, the lack of defected physical product samples results in poor diversity of generated defect images which is further propagating to the training procedure. Moreover, those methods lack controllability since the only way to control them is through input images, not parameters. Next, those methods cannot generate features they have not seen, including different imaging and illumination conditions. Finally, the results are images and a substantial amount of work is required for incorporating them into 3D virtual environments.

6.3 Requirements

Procedural geometry modeling approaches have been widely developed and adopted in the film and game industry. One reason for this is the modular and algorithmic description of objects, which allows significant content creation flexibility. The resulting parameterized models can be used for rapid object creation by simply tuning the parameters. Unfortunately, existing general-purpose procedural modeling tools (e.g., Houdini) are not intuitive to inspection planning experts or suitable for integration with inspection planning. A promising solution lies in specialized parameterized models for defect creation, stemming from a collaboration between computer graphics experts and domain experts. The resulting defecting models should offer parameters intuitive to an inspection planning expert, who can use them to create defected product instances in a virtual environment. Based on the discussion with a domain expert, defecting models should:

R10 Be configurable by a set of predetermined parameters

R11 Ensure required defect diversity

R12 Require only intuitive input parameters

R13 Be applicable to arbitrary free-form product geometry

R14 Provide means for automatic defect annotation

R15 Be usable in image synthesis environment

R16 Be usable in interactive inspection planning environment

Defecting models must provide parameters for controllable creation of arbitrary amount of different defected geometries (**R10**). Provided parameters must be configurable so that the creation of diversified defects is ensured (**R11**). Parameters must be usable by an inspection planning expert who is not necessarily a computer graphics expert (**R12**). Defecting models must perform correctly regardless of product geometry shape (**R13**). Models must generate information needed for defect annotation in the synthesized images (**R14**). Created defected objects must be ready for image synthesis and therefore defecting models must be compatible with image synthesis environments (**R15**). Finally, defecting models must be usable in an interactive inspection planning environment for image acquisition, hardware configuration and planning of its placement (**R16**).

6.4 Methods

Two parameterized models based on procedural geometry methods for the controllable creation of dent and scratch defects are presented. Presented defect creation models are based on the following workflow (Fig. 6.1):

(1) Generate defect positions.

(2) Generate defecting tool geometry for dent and scratch defects for each position.

(3) Create defect shell geometry.

(4) Imprint the defecting tool geometry into the product geometry surface.

Step (4) consists of CSG Boolean difference [216] between product object geometry and defecting tool geometries.

Defect positioning methods

Defect positioning methods define the distribution of points on object surface which are in later steps used for placing defecting tool. Thus, they define where the defecting will take place.

Manual positioning (Algorithm 5). User picks screen-space coordinates P_{screen} which are used for ray-casting with mesh M. Found intersections are used as defecting positions P. Although manual, this method can help for precise interactive defecting by inspection planning experts.

Random placement (Fig. 6.2a and Algorithm 6). First, for each face of the given mesh M, the number of face samples $n_{face\_samples}$ is calculated based on the face area and user-defined density $d > 0.0$. This way, larger faces will have more samples and smaller faces will have fewer samples.

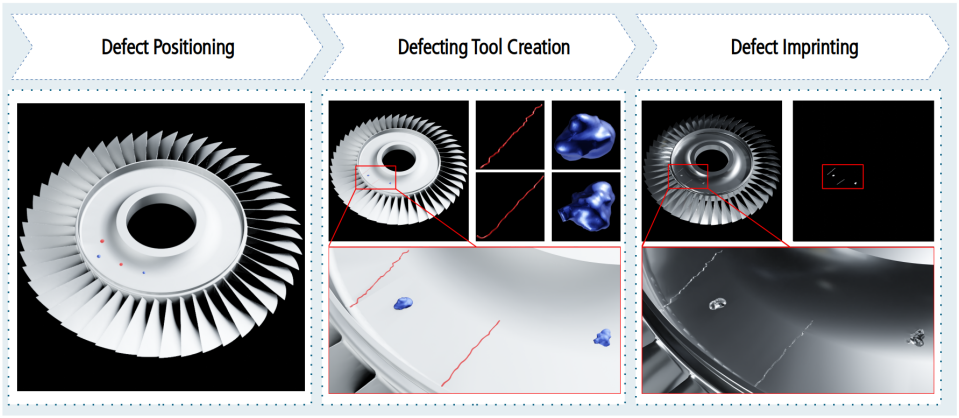


Figure 6.1: Defected geometry creation workflow. First, defect positions are generated over the surface. Second, for each generated position, a unique dent (blue) or scratch (red) defecting tool is created. Finally, the created defecting tools are imprinted into the surface.

Algorithm 5 Manual placement

Require: M, P_{screen}
Ensure: P
  for P_{screen_i} *in* P_{screen} **do**
    $P \leftarrow raycast(M, P_{screen\_i})$

Higher density enables generating more samples on a face. Next, for each face, random barycentric sampling is performed $n_{face\_samples}$ times. Finally, when all faces are sampled (P_{all}), user-defined amount n of random samples are selected from all samples P_{all} as defecting positions P. This method is inspired by Blender's [150] point distribution geometry node.

Algorithm 6 Random placement

Require: M, n, d
Ensure: P ▷ Positions
  $P_{all} \leftarrow [\,]$
  for f_i *in* $M.faces$ **do**
    $n_{face\_samples} \leftarrow \lceil area(f_i) \times d \rceil$
    $P_{all}.add(sample(f_i, n_{face\_samples}))$
  $P \leftarrow pick\_n\_random(P_{all}, n)$

Localized placement (Fig. 6.2b and Algorithm 7). First, the bounding volume hierarchy (BVH) acceleration data structure of object mesh M is constructed. Next, n points are randomly sampled within the user-defined cube volume of size S and centered at C. Finally, BVH is used to find points

on the object mesh which are closest to all of the sampled points V_{points}. The found points are further used as defect positions P.

Algorithm 7 Localized placement

Require: M, n, S, C
Ensure: P ▷ Positions
  $bvh \leftarrow$BVH(M)
  $V_{points} \leftarrow sample\_cube(C, S, n)$
  $P \leftarrow bvh.nearest(V_{points})$

Edge placement (Fig. 6.2c and Algorithm 8). First, for each edge of the given mesh M, an angle θ between faces sharing that edge is calculated. Next, if θ is within the user-specified range ϕ_{min} and ϕ_{max}, then a number of points on that edge are sampled randomly and stored (P_{all}). The number of edge samples is calculated based on edge length and user-specified density parameter $d > 0.0$. Finally, n random samples are taken from P_{all} as defecting positions P.

Algorithm 8 Edge placement

Require: $M, n, \phi_{min}, \phi_{max}, d$
Ensure: P ▷ Positions
  $P_{all} \leftarrow [\,]$
  for e_i *in* $M.edges$ **do**
    $\theta \leftarrow neighbouring\_faces\_angle(e_i)$
    if $\theta > \phi_{min}$ *and* $\theta < \phi_{max}$ **then**
      $n_{edge\_samples} \leftarrow \lceil len(e_i) \times d \rceil$
      $P_{all}.add(sample(e_i, n_{edge\_samples}))$
  $P \leftarrow pick\_n\_random(P_{all}, n)$

Cluster placement (Fig. 6.2d and Algorithm 9). Methods (1), (2) and (3) can be extended so that for each generated sample, a cluster of samples is generated in an arbitrary neighbourhood. First, BVH for fast spatial queries is constructed using a given mesh M. Then, for each given starting point $p_i \in P_{start}$ a cube of size S and center p_i is sampled for $n_{cluster}$ points and stored (P_{all}). Finally, for all sampled points P_{all}, the BVH was used to find the closest points P on mesh which are later used as defecting positions.

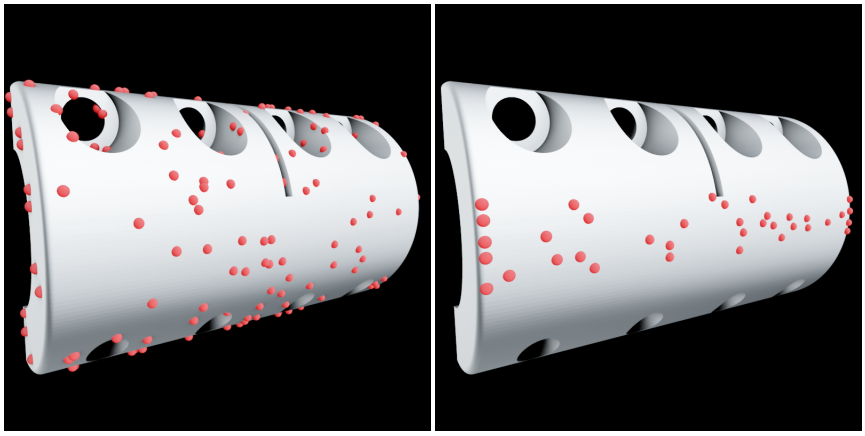
Generating denting tool geometry

For each generated dent position, the following parameterized models for creating denting tool geometries can be used:

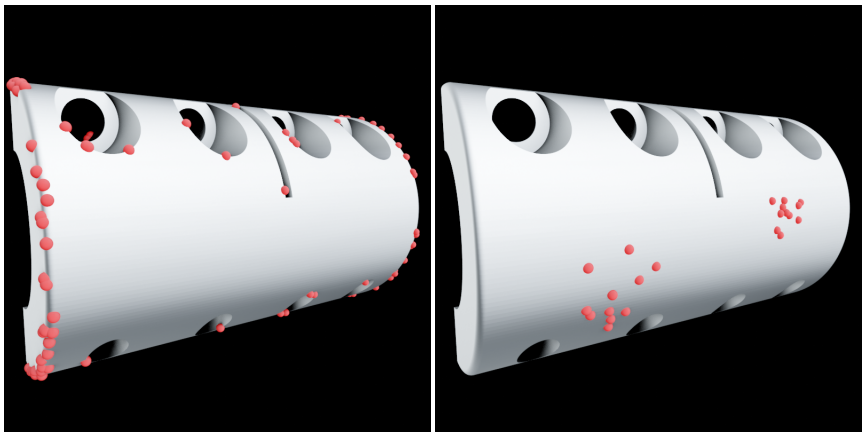
Spherical denting tool (Fig. 6.3a and Algorithm 10). First, triangulated sphere mesh with n_{verts} vertices is created. Next, scaling and rotation are performed using random scaling factors and rotation angles from

Algorithm 9 Cluster placement

Require: $M, P_{start}, n_{cluster}, S$
Ensure: P ▷ Positions
  $bvh \leftarrow$BVH(M)
  for $p_i \in P_{start}$ **do**
    $P.add(sample\_cube(p_i, S, n_{cluster}))$
  $P \leftarrow bvh.nearest(P)$

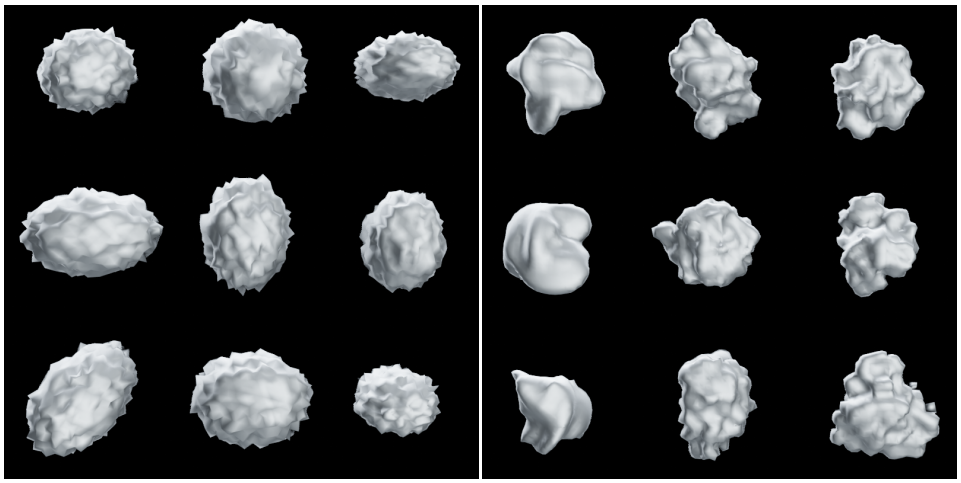


(a) Random placement. (b) Localized placement.



(c) Edge placement. (d) Cluster placement.

Figure 6.2: Defect positioning algorithms.



(a) Spherical denting tools. (b) Clustered denting tools.

Figure 6.3: Examples of denting tool geometries.

user-specified ranges s_{range} and r_{range} respectively. Finally, displacement of vertices in normal direction using noise is performed, parameterized by strength $d_{strength}$ and frequency d_{freq}.

Clustered denting tool (Fig. 6.3b and Algorithm 11). First, points inside of a cube volume with size C_{size} are sampled randomly. Next, on each sampled point, randomly scaled and rotated elements (e.g., sphere or cube mesh) are placed so they are partially overlapping. Further, the created elements are merged into one object using voxel remeshing. Finally, resulting mesh can be scaled and rotated by user-specified ranges s_{range} and r_{range}. Additionally, vertex displacement in normal direction can be performed controlled by noise with user-specified frequency d_{freq} and strength $d_{strength}$.

Algorithm 10 Spherical denting tool creation

Require: $n_{verts}, s_{range}, r_{range}, d_{strength}, d_{freq}$
Ensure: D ▷ Denting tool geometry
$D \leftarrow create\_sphere\_mesh(n_{verts})$
$D \leftarrow scale(D, s_{range})$
$D \leftarrow rotate(D, r_{range})$
$D \leftarrow vertex\_displace(D, d_{strength}, d_{freq})$

Generating scratching tool geometry

Each generated scratch defect position is used as starting point for building scratch keypoints using mesh walking. Next, scratch keypoints are used for

Algorithm 11 Clustered denting tool creation

Require: $C_{size}, s_{range}, r_{range}, d_{freq}, d_{strength}$
Ensure: D ▷ Denting tool geometry

```
C_positions ← sample_cube_volume(C_size)
C_elements ← []
for p in C_positions do
    e ← create_mesh("sphere"|"cube")
    e_scale ← random_vector(C_size)
    e_rotate ← random_scalar(0, 2π)
    e ← transform(p, e_scale, e_rotate)
    C_elements.add(e)
D ← remesh(C_elements)
D ← rotate(D, r_range)
D ← scale(D, s_range)
D ← vertex_displace(D, d_freq, d_strength)
```

building solid geometry which represents a scratching tool. Finally, scratching tool geometry is refined (e.g., vertex displacement).

Scratch keypoints roughly define scratching tool geometry (Fig. 6.4) and are generated as described in Algorithm 12. First, a BVH acceleration structure is built using the given mesh M. Next, path ending point P_{end} is generated using BVH, so it lies in user-defined distance range R from starting point P_{start}. Finally, path keypoints are generated iteratively using BVH neighbourhood search from starting point to ending point with user-defined distance S.

Algorithm 12 Generating scratch keypoints

Require: M, P_{start}, R, S
Ensure: $P_{keypoints}$ ▷ Scratch keypoints

```
P_keypoints ← [P_start]
bvh ← BVH(M)
P_end ← farthest(bvh.nearest(P_start, R))
tmp ← P_start
while dist(tmp, P_end) > 0 do
    dir ← normalize(P_end − tmp)
    next = tmp + dir ∗ S
    next = BVH.nearest(next)
    P_keypoints ← [P_keypoints, next]
    tmp ← next
```

Generated scratch keypoints are further refined (e.g., displaced) and used for constructing solid geometry representing scratching tool (Fig. 6.4). Solid

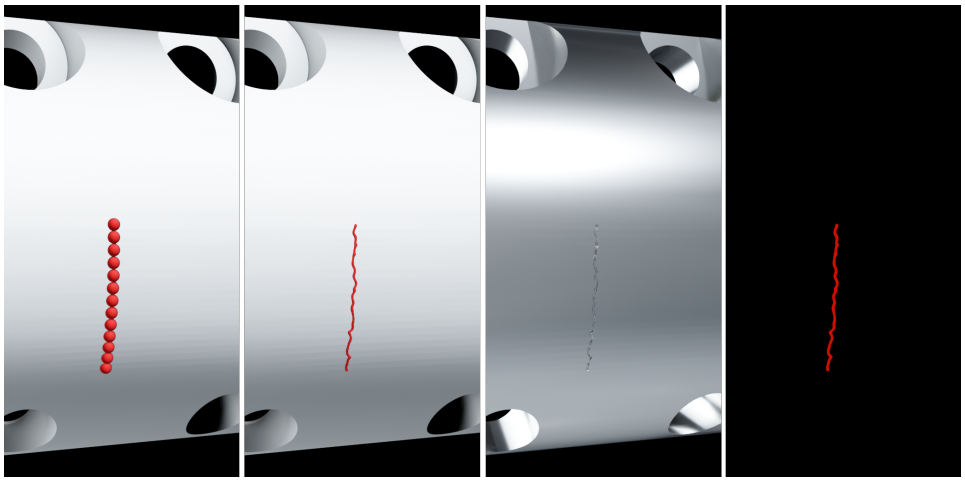


Figure 6.4: Scratch keypoints, scratching tool geometry, scratch render, scratch mask (annotation).

geometry can be built using the Bezier curve or metaball object. Bezier curve is built using keypoints as Bezier control points. Next, the curve is thickened and transformed into the mesh. Finally, mesh vertices can be additionally displaced in vertex normal direction using noise. Alternatively, the metaball object is constructed from generated keypoints by firstly interpolating additional points between generated path keypoints. Next, on each point, a metaball object is placed. As metaballs are analytically defined, they "melt" in one another when overlapping, forming continuous solid geometry. Finally, the metaball object is converted to mesh and mesh vertices can be further displaced in the vertex normal direction using noise.

Generating defect shell geometry

Defect shell geometry (Fig. 6.5) is constructed using defecting tool geometry and CAD of the inspected object. First, the Boolean intersection between the CAD object and the defecting tool is calculated. Next, the defecting tool geometry is slightly downscaled (e.g., by a factor of 0.99). Finally, the Boolean difference between the scaled defecting tool and the intersected object from the previous step is calculated. The resulting geometry is a defect shell and is used for rendering defect annotations.

6.5 Results

The complete defecting workflow (Fig. 6.1) introduced in Section 6.4 was implemented using Blender [150] Python API and requires only Blender as a dependency and availability of a CAD model. The results of the defecting

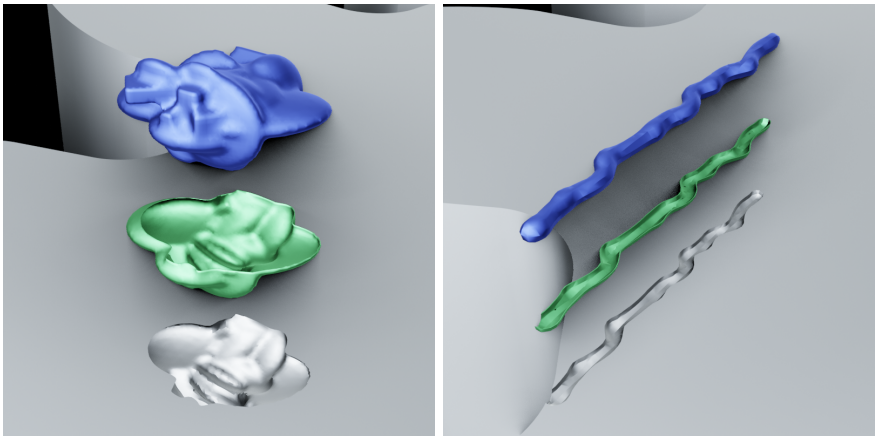
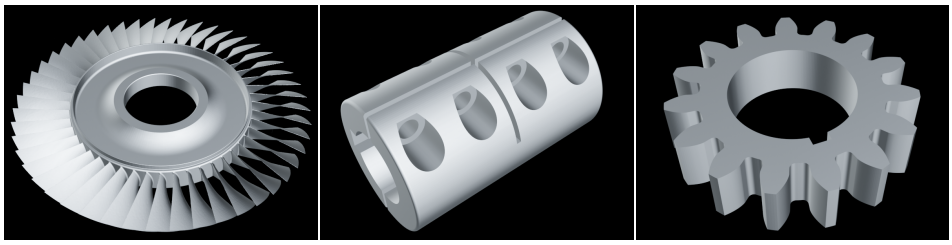


Figure 6.5: Dent (left) and scratch (right) defects with corresponding defect shells (green) and defecting tools (blue).



(a) Blisk (b) Clutch (c) Gear

Figure 6.6: Inspected object CAD models.

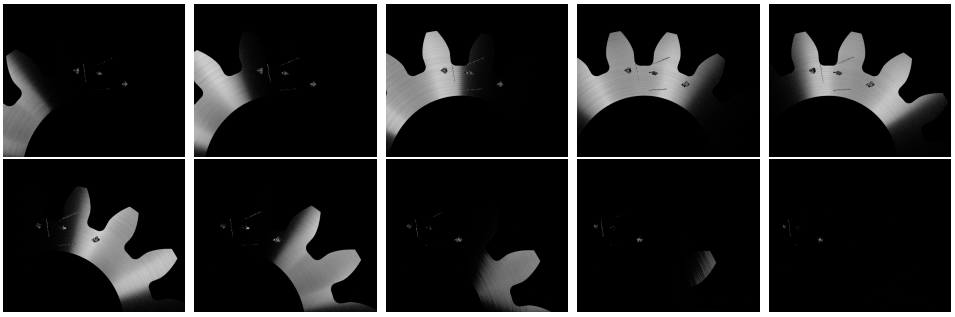


Figure 6.7: Scratches and dents visibility as view and illumination changes in direction of an arc above defects.

process were used in a virtual surface inspection environment containing the inspected object, illumination and camera described in Chapter 4. The 3D scene is rendered using appleseed [184], a physically-based, path-tracing rendering engine.

Inspected objects used in virtual environment are: blisk, clutch and gear (Fig. 6.6). All mesh objects have uniformly sized and distributed triangles across the surface. Blisk, clutch and gear meshes are composed of 200K, 80K and 11K triangles respectively. Blisk and clutch objects are CAD models representing real inspection parts whereas the gear CAD model is created to resemble real inspected parts. Material is defined by appleseed's metal BRDF model and OSL [185] texture models introduced in Chapter 5.

The illumination source is defined by torus-shaped mesh geometry and OSL's diffuse-emission material. Its position is relative to the camera, meaning that it is mounted on the camera and its position transformation follows the camera transformation. The environment is black and non-emissive.

The camera is defined by appleseed's pinhole camera model: pixel size 0.0069 mm, focal length 12.93 mm and resolution 1224×1025 pixels. The parameters correlate with real camera parameters.

Fig. 6.7 shows defected gear geometry illuminated from different angles. Comparison of the real and simulated clutch object containing defects is given in Fig. 6.8. A close-up comparison of the real and simulated dent and scratch defects on clutch and gear objects is given in Fig. 6.9. Defect shell geometry (Section 6.4) was used for rendering annotations. First, a 3D scene containing inspected CAD object with assigned black, non-emissive material and defect shell geometry with assigned white emissive material is created. Next, sources of illumination were disabled and environment illumination was set to black, non-emissive. Finally, rendering the described 3D scene resulted in defect annotations as can be seen in Fig. 6.10 for the defected blisk object.

6.6 Discussion

Presented defecting models are controllable by a set of parameters, as discussed in Section 6.4, satisfying requirement **R10**. Defecting model parameters are provided in ranges that can be sampled to obtain a space of diversified defects over the object surface (see e.g., Fig. 6.10), satisfying requirement **R11**. Defecting model parameters are defining defects in terms of their spatial distribution and shape, attenuating the need for computer graphics knowledge (see Section 6.4), making the presented models more intuitive to inspection planning expert, partly satisfying requirement **R12**. Discussion with experts regarding what makes defecting models usage even more intuitive is left for future work. Defecting models can be used on complex, free-form geometry (e.g., Fig. 6.8), satisfying requirement **R13**. As

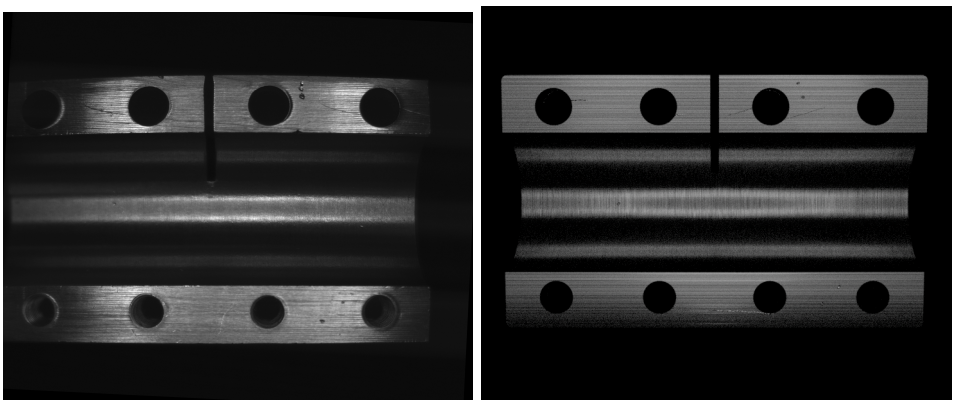


Figure 6.8: Comparison of real (left) and simulated (right) defected clutch object.

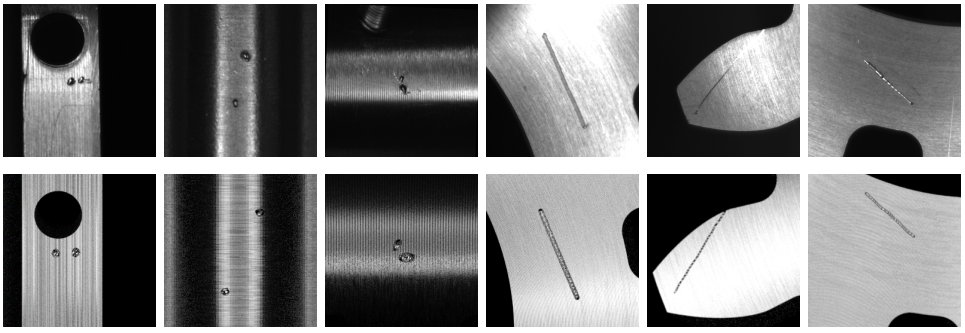


Figure 6.9: Comparison of real (top) and synthetic (bottom) dent and scratch defects.

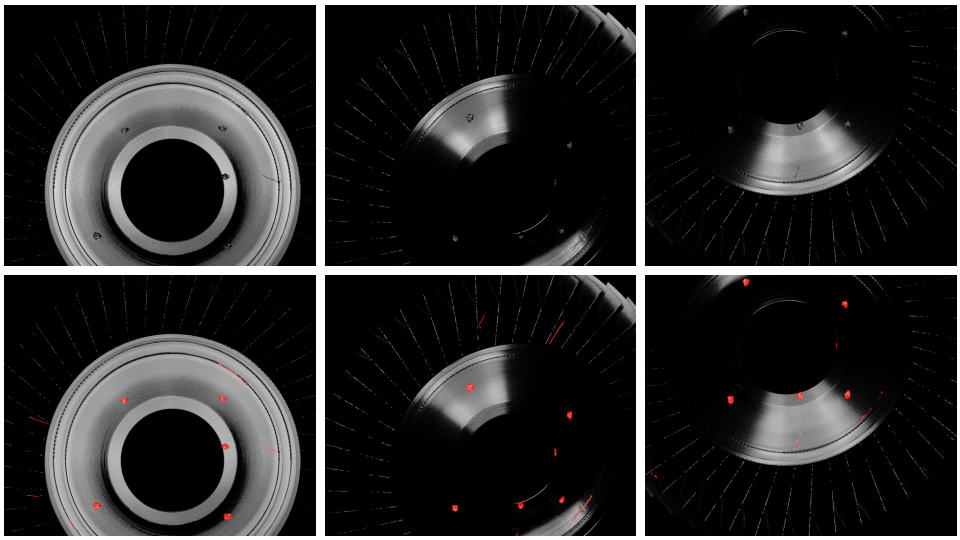


Figure 6.10: Defected instances of blisk object (above) and annotations (below).

discussed in Section 6.4 and visible in Fig. 6.10, automated, pixel-precise defect annotation is supported, satisfying requirement **R14**. Presented defecting models were used in existing image synthesis (Chapter 4) and inspection environment (Gospodnetic et al. [6]) to produce resulting images, therefore, satisfying requirements **R15** and **R16**.

Fig. 6.7 shows defect visibility under changing illumination and viewing positions and thus the importance of geometrically imprinted defects. Changes in illumination and viewing configuration cause different parts of defects to be visible. Especially interesting cases for surface inspection planning are grazing angles, where only the defect is visible and the rest of the surface is invisible. As defect shapes are often geometrically complex they can trap or redirect light in a complex manner. This kind of light-defect interaction can be only achieved with geometrical modeling of defects as opposed to texture and image-based defect modeling discussed in Section 6.2. Therefore, the presented methods serve inspection planning expert for assessing defect visibility of acquisition hardware configuration, without the need for physical samples.

Presented models enable the creation of defected products with diverse dent and scratch defect shapes which are variously distributed over the surface (Fig. 6.10). The complexity of created defect shapes can be seen in Fig. 6.9 where a close-up of dent and scratch defects is presented. Furthermore, comparison of real and simulated defects (Fig. 6.8 and 6.9) shows that simulated defects are representative of real defects to a high degree. This example clearly emphasizes the importance of geometrical defects for visual surface inspection planning.

Presented parameterized defecting models enable automated creation of arbitrary amount of diversified defected product geometries. Furthermore, a method for defect annotations creation (Section 6.4) enabling the generation of training-ready synthetic datasets was presented. Finally, presented defecting models do not depend on existing defected product samples which is the case with image-based defecting approaches (Section 6.2).

Procedural methods for the creation of parameterized models opened new venues for collaboration between computer graphics experts and surface metrology experts. Our aim is to further incorporate domain knowledge into defecting models. Thus, further research regarding defect placement distribution is required. For example, defect locations can be weighted on particular object areas due to object geometrical features and manufacturing process. Next, further research is to be conveyed regarding defect shapes and their imprinting into the object surface. To explain, defecting areas might contain surface mass redistribution causing "wrinkled" deformation. Also, defected areas can exhibit altered material properties besides geometrical deformation.

6.7 Conclusion

In this work, parameterized models for the controllable and automated creation of defected product geometries are presented. The resulting geometries contain geometrically imprinted defects which ensure correct and realistic light interaction and visibility. Presented defecting models are usable in a virtual environment by an inspection planning expert to help assess defect visibility for a given inspection hardware configuration. Furthermore, presented defecting models enable representative and diversified defected product samples creation and thus offer a solution to the lack of defected physical product samples. Finally, automated creation of an arbitrary amount of defected product images with precise annotations is available giving a way to training-ready datasets.

With the presented approach, now it is possible to obtain the required amount of representative defected products needed for both VSI planning and the development of machine vision algorithms for defect detection. Furthermore, the presented approach opens a new venue for domain knowledge integration into computer graphics modeling, which calls for further collaboration between computer graphics and surface metrology experts. This would lead towards a much needed library of parameterized defect creation models for virtual environments. Such libraries would serve for further ML models advancements for defect detection.

Core references

Lovro Bosnar, Hans Hagen and Petra Gospodnetic, "Procedural defect modeling for virtual surface inspection environments", 2023. IEEE Computer Graphics and Applications, 43(2), 13-22., *doi:* 10.1109/MCG.2023.3243276

Part III

Applications

Chapter 7

Synthetic Data for Defect Segmentation on Complex Metal Surfaces

Metal defect segmentation poses a great challenge for automated inspection systems due to the complex light reflection from the surface and lack of training data. In this work we introduce a real and synthetic defect segmentation dataset pair for multi-view inspection of a metal clutch part to overcome data shortage. Synthetic dataset was generated using image synthesis methods coupled with parameterized texture and defect modeling introduced in Chapters 4, 5 and 6. Model pre-training on our synthetic dataset was compared to similar inspection datasets in the literature. Two techniques are presented to increase model training efficiency and prediction coverage in darker areas of the image. Results were collected over three popular segmentation architectures to confirm superior effectiveness of synthetic data and unveil various challenges of multi-view inspection.

For the sake of completeness, this chapter includes work done by PhD student Juraj Fulir who (1) characterized and described real-world physical sample (clutch object), its texture, defects and performed acquisition (Section 7.3), (2) performed defect segmentation and experimental evaluation (Sections 7.3 and 7.4).

7.1 Introduction

Surface inspection is a common task of automated visual inspection systems, where defects are anomalies appearing on the surface of the product, resulting from production chain error (e.g., scratch, bump, pitting) [222]. While automated surface inspection systems introduce benefits such as faster inspection process and reduction of human error, they require consistent acquisition conditions. The acquisition conditions, together with detection algo-

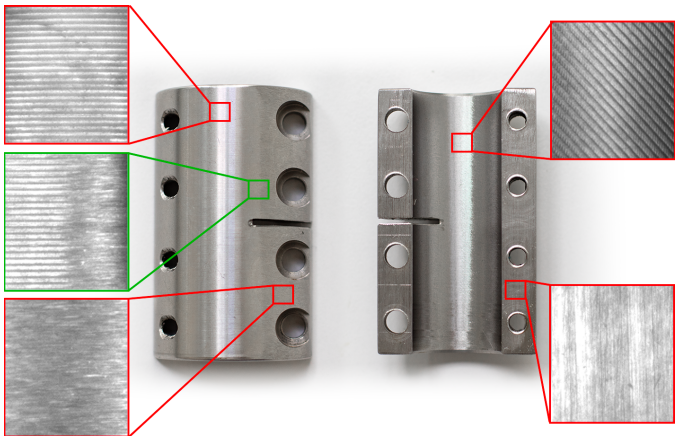


Figure 7.1: The examined clutch object from outside (left) and inside (right). The object contains four distinct texture patterns (red) and a transition area (green).

rithms, are always customized for specific inspection tasks. This makes the systems extremely rigid. Experts planning the acquisition hardware setup in an inspection system must ensure that the whole surface of the inspected product is illuminated and captured in a way such that all possible defects can be successfully detected. The task may sound simple in principle, however the appearance of defects varies drastically and is largely affected by the location of the defect relative to illumination sources and camera (fig. 7.2). This problem becomes especially noticeable on geometrically complex and reflective metallic surfaces. Additionally, defect visibility can be obscured by the surrounding surface texture, which vary locally (fig. 7.1) and between products.

Designing a robust inspection system requires defect samples which are diverse enough to provide a complete understanding of all the possible defect characteristics and occurrences on the production line. A sufficient dataset will thus require a large number of physical samples, which might be challenging to obtain since some defects appear more frequently than the others and appearance within a single class of defects can vary greatly. The challenge increases for premium products fabricated in low volumes. While traditional image processing algorithms can be developed with considerably smaller amount of defected samples, cases with high variation of defect characteristics significantly complicates their development and maintenance. Machine learning approaches can circumvent these shortcomings by relying on automatic extraction of robust features from large amounts of diverse data. However, in low data scenarios they are prone to overfitting.

Usage of synthetic data to circumvent the data shortage has gained traction recently in machine vision [218, 223–227]. Mainly because it provides

a way to generate arbitrary amount of diverse annotated training data, including edge-case scenarios which are difficult to obtain in real production. However, **there is a lack of studies which investigate the suitability and advantages of using custom designed synthetic data for industrial quality inspection**.

We summarize our contributions as following:

- We introduce a dual dataset, consisting of real and synthetic equivalent, for the domain of multi-view inspection of a complex metal object to expand the existing literature on synthetic data for industrial applications.
- We compare the effectiveness of our synthetic dataset to alternative metal inspection datasets in the literature, to confirm that a custom designed synthetic data is superior in the low-data scenario.
- We introduce intensity biased cropping mechanism to increase model training performance in this domain.
- We introduce exposure stacking to increase model response in darker regions and discuss its effect on surface coverage in inspection.
- Finally, we identify the unique shortcomings of applying synthetic data in this domain and offer research directions for overcoming them.

7.2 Related Work and State of the Art

7.2.1 Defect Recognition

Defects are the results of anomalous events in the production chain which inflict deviations from product's intended design, function, or appearance. Defects can come in various forms, with several classifications present in the literature [222,228,229], however, what is and what is not considered to be a defect is always application-specific. For visual inspection we distinguish between defects which are visible by observing the object with non-penetrating light interaction [222,230], and defects that are below the observable surface and should be inspected using a material penetrating medium [230–232]. In this work we focus on the first kind, more specifically, macroscopic surface defects such as dents and scratches [222].

Recognition approaches Defect recognition is a process of identifying a defect and its characteristics. There is a number of traditional (non-learning) approaches to application specific defect recognition [228, 233]. However, these methods rely on manual design and, when presented with changes in inspection setup, require redesign which leads to an increase in complexity

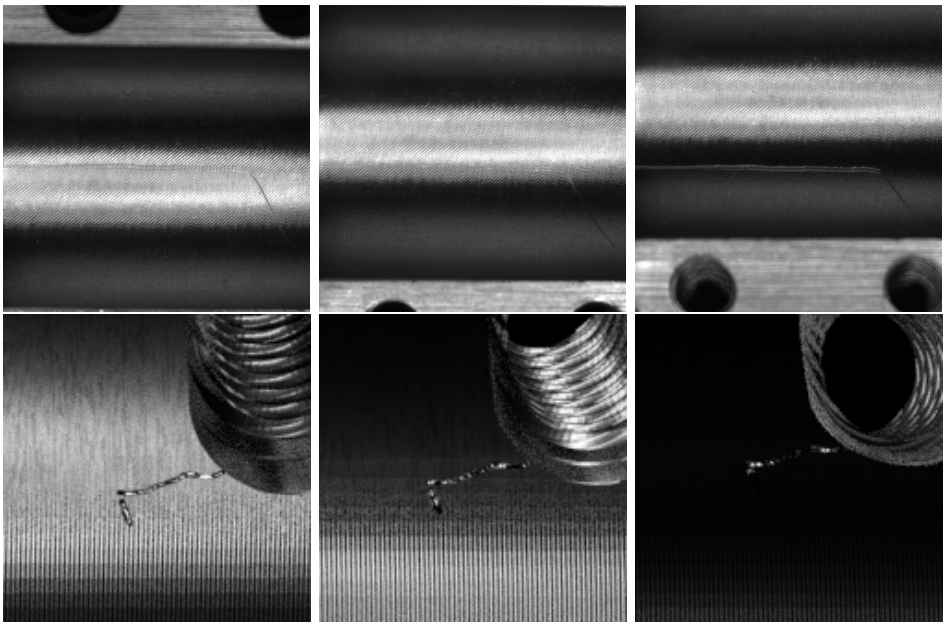


Figure 7.2: Appearance inconsistencies of scratches on a curved surface when acquired under different angles, as seen in real (top) and synthetic (bottom) data. Defect appearance changes in terms of its shape and contrast due to illumination and the surrounding surface texture. Notice how the defect part gets obscured by the texture in the upper middle image.

and operating cost. Therefore, recent research aims largely at learning based approaches where defect recognition models are trained under supervision with labeled data. The approaches can be aimed at defect detection [229, 234], defect segmentation [229, 235–237] or image classification [229, 236]. In all cases, the main problem is the high cost of obtaining large amounts of labeled data. This may increase difficulty of developing models which successfully generalize to production.

Labeling for both detection and segmentation is time consuming. Therefore, efforts have been made to use methods which rely on faster training sets annotation approaches, such as image classification [229]. There, segmentation is achieved using the class-activation map (CAM) technique [238]. CAMs tend to produce very localized predictions, which are useful for defect segmentation as defects are often localized, such as defects in LED chips [239]. Božić et al. [236] mixes pixel and image level labels to increase the effective dataset size at a lower annotation cost. In cases where defected samples are unavailable or are in too small quantities for supervised training, anomaly detection can be used on solely the correct samples for training. During inference, the reconstruction of an input image is compared to the original or extracted features are compared to memorized features of correct

samples to detect outliers. It has been employed for defect segmentation over a variety of objects as presented in [240–243]. We focus on the case where training data is scarce, making the aforementioned methods unsuitable.

Datasets Various datasets exist for defect recognition tasks in metal [228, 229, 244]. Most available datasets focus on inspection of hot-rolled steel [245–248] which is a planar surface with various defects. Other present shapes are curved pads [235, 236, 249], pipes [250] or rails [251]. More complex surfaces are present in [252] in form of a ball screw driver with a multi-view setting through single-axis rotation. Anomaly detection datasets [243, 253] contain complex metal objects, however they reduce the problem to single-view inspection with a fixed top-down view. In [237], authors perform a similar top-down acquisition with varying illumination angles. In contrast, our dataset represents a complex geometry with highly specular and anisotropic surface in the multi-view setup in a dark environment.

Inspection of complex surfaces Defect recognition is tightly coupled with inspection planning process, which determines the image acquisition setup (i.e. camera and illumination position). This process is currently performed by experts based on physical tests and experience. A semi-automated inspection planning pipeline for virtual design and verification of inspection plans was introduced (Chapter 4 and [3]). That work allows coverage evaluation of any object geometry, regardless of its geometrical complexity. In this work we rely on their methods to create inspection plans for both real and synthetic data.

7.2.2 Synthetic Data Generation

Image synthesis can benefit data preparation by providing more control over its content and diversity, speeding up the process and reducing its costs. Additionally, it provides insight into the expected inspection coverage and results. So far it was employed in many forms to a wide range of machine vision tasks [214] in order to produce balanced datasets for machine learning.

Generative models A straight-forward way to generate defected samples from correct real images is by synthesizing an image of a defect and embedding it in a correct real image. The cheapest approach is by manually designing generative models which produce 2D patches of defects [254]. Albeit a controllable and versatile technique, the defects are modeled as 2D patches and can not correctly model the light response of specular defects observed from multiple viewpoints. This introduces a bias towards the subset of modeled defect appearances with inaccurate light response. A more popular approach is to automatically learn the generative model using generative-adversarial networks (GAN), where two models are jointly

trained in adversarial setup on weakly-annotated real data [229, 255, 256] with control over the spatial properties, category and style of defects. These approaches demonstrate great improvements in the defect recognition tasks, however they can not introduce data representing edge-case scenarios or guarantee generation of correct data. The second requirement is particularly difficult to obtain in the multi-view setup due to the complex specular defect appearance. Additionally, extending the supported set of defect types or variations requires retraining on new observed data which does not guarantee the retention of appearance quality in previously supported defect types.

Computer graphics Leveraging computer graphics for data generation provides a versatile, controllable and reliable tool for generating large quantities of data with the support for generation of scenario-specific variations. It has proven its usefulness across various computer vision tasks [257]. A popular example is traffic scene recognition where the synthetic datasets are commonly paired with real datasets [227] to complement the scenarios missing from the real data. In situations where manual data annotation is intractable, it offers an invaluable source of annotated data such as in the many-keypoint tracking task [226].

In defect recognition domain, available synthetic datasets are sparse. The DAGM dataset [258] consists of generic artificial textures and defects with no specific application in mind and is often used as a baseline benchmark. In [259], authors use simple noise transformations for color and vertex displacement to generate a labeled synthetic dataset for defect detection over steel plates. The MIAD dataset [224] is a product maintenance dataset for anomaly detection with various outdoor scenarios, including welding defects of a steel pipe. The recent CAD2Render toolkit [223] produced the DIMO dataset [260] by relying on photo-realistic rendering. It goes a bit further by using procedurally generated defects which are applied to the object surface to simulate rust and scratches. However, it focuses on assembly inspection and object pose estimation without specific control over the shape and locations of defects, reducing its usefulness for defect inspection.

In all of the above mentioned cases, the main focus is on simulating macroscopic features such as the object shape or surface color, disregarding the evaluation of the correct light response from micro-scale structures of the surface texture or the defect geometry. This is not sufficient in cases when the surface is observed from multiple viewpoints at higher resolution, reducing their usability for defect recognition. In chapters 5 and 6, methods have been introduced for the generation of procedural textures and procedural defects, capable of approximating various industrial surfaces with a high degree of realism and control. The synthetic defecting methods have already been employed in [218] for defect segmentation in endoscopic images

of a turbocharger.

Transfer learning Transfer learning techniques aim to align the problem domain between different data sources. A number of techniques is at hand, depending on the task and data availability [261]. These methods are suitable for use with synthetic data since the synthetic data introduces various approximations of the real world appearance, thus creating a domain shift.

Domain adaptation exploits the knowledge obtained from the source data to align the model towards the target data. The most common approach is by initializing the training procedure with model parameters pre-trained on a larger source dataset [214, 227, 261, 262]. The model can be adapted entirely [263] or partially [242]. However, some research suggests that similarity between the two domains results in better performance [261, 263].

Domain randomization [264] is a technique which enlarges the variance of the source domain to increase the chance of covering the target domain, while making it possible for the model to learn more robust features. It is commonly used in synthetic data since the environment can be easily manipulated [214, 227]. By parameterizing the surface texture and defect geometry as procedural functions as discussed in 5 and Chapters 6, we can generate a variety of data that can cover all plausible possibilities within the specified ranges. Note that some parameters, such as perspective distortion, rotation or flipping of the image, are commonly randomized through train-time augmentation removing the need for their rendering.

Following Wood et al. [226] instead of using domain adaptation to reduce the domain gap, we rely on increasing the realism of synthetic data. However, we do incorporate domain randomization of surface appearance and background to produce a variety of realistic samples.

7.3 The Clutch Dataset

In this work we introduce and publish[1] the dual dataset composed from real data and its synthetic equivalent. It is a versatile dataset which can be utilized for multiple tasks such as image classification, defect segmentation and detection in supervised, unsupervised or weakly-supervised approaches. In this work we focus on binary defect segmentation in order to evaluate approaches for dataset preparation, machine learning techniques specialized for this domain and possible imperfections of synthetic data which must be taken into account.

[1] `https://owncloud.fraunhofer.de/index.php/s/mtr1FzERutdOrXi`

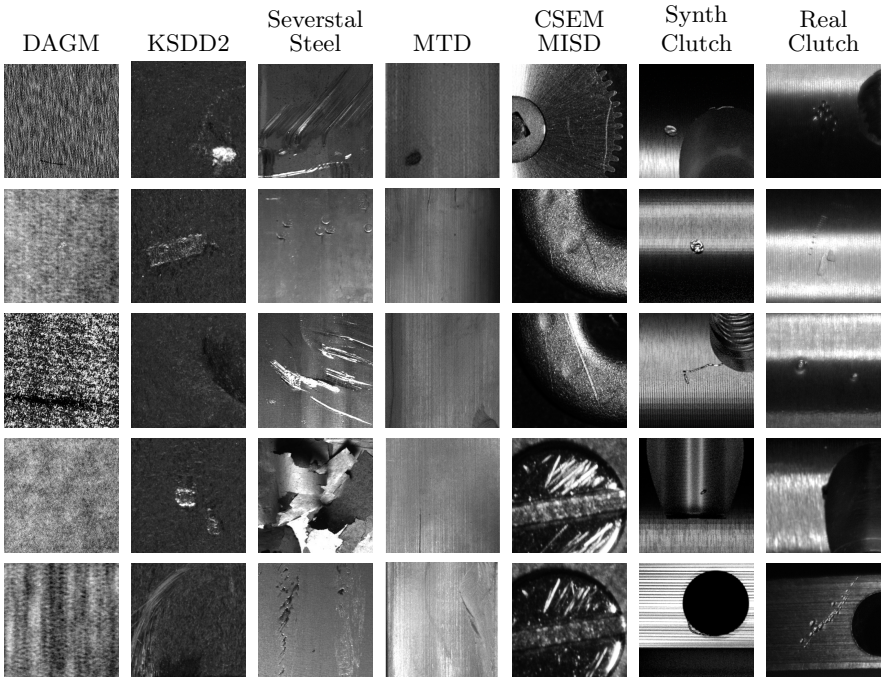


Figure 7.3: Texture and defect examples extracted from datasets used in this work. Best viewed digitally.

7.3.1 Object Description

The dataset contains a part of a clutch, shown in fig. 7.1. The clutch is an aluminum object consisting of two halves, produced using turning and milling with additional brushing to remove material extrusions introduced from drilling. The flat and curved surfaces, holes and details such as screw threads or beveled edges increase the geometrical complexity of the object. Different machining and processing operations throughout the part production introduce four distinct surface textures displaying patterns with more or less prominent periodicity.

Since the real defects are often a proprietary information, the clutch object is used as a case study and the defects were introduced manually to resemble typical defects appearing in production lines. The defects include various scratches and dents depicted in fig. 7.3.

7.3.2 Real Data Acquisition

The RealClutch acquisition setup consists of a robot manipulator, matrix grayscale camera with a diffuse ring light mounted around it and the acquisition table. The manipulator is used to position the camera and the illumination into predefined viewpoints. The acquisition table is a flat sur-

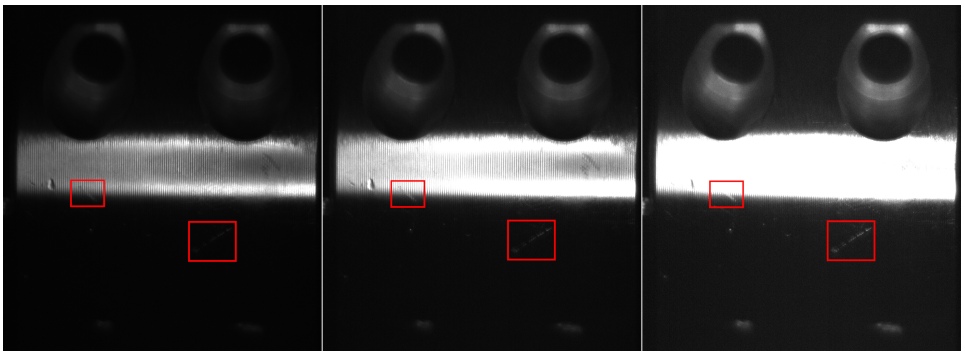


Figure 7.4: Increasing image exposure reveals the dark, but also overexposes the bright parts of the object. Notice the appearance change of the lower right scratch and the middle left scratch.

face covered in diffuse black velvet and the inspected object is placed on it. The viewpoints were arranged manually using V-POI[2] [7] in a way that covers the inspected surfaces with overlaps [8]. For the purpose of this work, the viewpoints have been created with significant overlap in order to examine defect behavior from multiple acquisition angles. Before the acquisition, hand-to-eye calibration was performed as discussed in Chapter 4, however, slight acquisition offsets are still present due to manual object placement on the acquisition table. The collected images were manually annotated using *labelme* [265] with an extension which allows enhancement of the defect visibility (fig. 7.4) by manually adjusting the image exposure using: $f(x) = x \cdot 2^{\alpha}$, with $\alpha \in \{0, 1, 2\}$. The polygonal annotations were finally rasterized into image masks used for model training and validation.

7.3.3 Synthetic Data Generation

The SynthClutch dataset has been designed and generated using the methods presented in Chapters 4, 5 and 6. The process requires a 3D model of the object and consists of four steps: inspection planning, defect modeling, texture definition and dataset generation. In the inspection planning step, view and illumination points are positioned in the space relative to the object. The viewpoints contain camera parameters such as resolution, focal length or focusing distance and illumination points contain the light geometry and intensity. For the purpose of this work, all the parameters correspond to the real setup, the lightpoint geometry has the ring light shape and is positioned around the viewpoint. The defects are modeled as dents and scratches, imprinted directly into the object geometry. They are defined using class specific parameter ranges, sampled to obtain desired shape variation. Dents are defined by their size, depth and elongation, while scratches

[2] `https://itwm.fraunhofer.de/v-poi`

are defined by their depth, length, and curving strength and frequency. The textures are modeled using procedural methods that perturb surface normals, with parameters adjusted to match the observed appearance of their respective counterparts across the real object (fig. 7.1). Finally, all aforementioned elements are combined and parameters are sampled to define a scene for photo-realistic rendering of the images and their corresponding defect masks, rendered using emissive material. As the real environment contains inter-reflections between the object and acquisition manipulator, we emulate this with a small amount of constant illumination. The background was dark with addition of uniform noise at train-time to resemble sensor dark shot noise.

The dataset contains object instances varying in defect shapes and texture which are then rendered according to the inspection plan. First, geometry instances are created with defects randomly generated and applied across the surface. The defect sizes were defined to be comparable to defects present in real samples and contain circular dents (diameter $0.2-2mm$) and scratches (diameter $0.05-0.3mm$). Additionally, we apply insignificant defects which do not contribute to the masks, but simulate minor irregularities present in the real data which may resemble defects but should be ignored. Next, the texture parameters are sampled within defined ranges centered around their previously optimized values. We randomize the surface roughness, normal perturbation strength and texture scale. Finally, each geometry instance is rendered with its corresponding texture parameters and stored in a structured way to simplify data loading. Reader is referred to the supplementary for a visual comparison of the two datasets.

7.4 Defect Segmentation on Complex Surfaces

Complex metal objects present a unique case for defect recognition due to the changes in surface appearance caused by reflectivity. This requires collection of large amounts of data to successfully train a model, however oftentimes this is not possible. In our case, only a small number of real samples is available real making it difficult to restrain models from overfitting, even with extensive data augmentation. Therefore, we look into using alternative data sources from similar domains and compare its use to a custom designed synthetic dataset. In both cases the real data is utilized only for fine-tuning and evaluation.

7.4.1 Utilizing Existing Planar Datasets

We first examine the transfer of features learned on available large datasets that represent similar domains. These datasets collect a large number of images with a variety of defect types observed over mostly planar surfaces. We restrict our selection to 5 datasets (fig. 7.3) based on similarities to

RealClutch: DAGM [258] for its genericness, KSDD2 [236] for its defect shapes, Severstal Steel [245] and MTD [249] for their material and defects, and CSEM-MISD [237] for its materials, defects and defect visibility changes.

7.4.2 Utilizing Custom Designed Synthetic Data

Synthetic data can be generated in arbitrary amounts with large diversity. However, it introduces a domain gap due to the approximations made during simulation such as texture geometry or material reflectance. We analyze the applicability of features learned on synthetic data with and without fine-tuning on the real data.

Intensity-biased random cropping Inspection images have larger resolution to increase surface coverage, which requires us to use random cropping during training. As large part of the image is in the dark, random cropping produces a large amount of crops that end up in the dark background regions. To remedy this, we bias the random cropping mechanism towards brighter regions to contain the object's surface. The input image is first binarized using a user-defined threshold to obtain an intensity mask. A random pixel from the mask is sampled and a crop window is centered around it. This technique guarantees that every crop will contain useful intensity values, while not completely ignoring the darker regions to prevent the possibility of detecting false positives in the acquisition scene background. The threshold was chosen heuristically from real data to ensure coverage of the regions containing manual annotations.

7.4.3 Enhancing Model Response in Dark Regions

When predicting on the acquired image, the model tends to respond only on well lit surfaces where the patterns have higher contrast, which reduces the detection coverage over the object surface. As described in section 7.3.2 annotators could increase image exposure to enhance the visibility of defect shapes in darker areas of the image, at the cost of overexposing some areas and increasing the image noise amplitude (fig. 7.4). To emulate this, we transform the acquired image using same exposure values as annotators to construct a channel-wise stack of transformed images alongside the original as inputs to the model. This allows our model to have simultaneous access to multiple exposure values and learn to respond to a much larger surface area.

7.5 Experimental Evaluation

7.5.1 Training Details

Training is implemented using PyTorch and for segmentation techniques we used easily accessible implementations of segmentation models: FCN [266] and DeepLabV3 [267] implementations from torchvision and U-Net [268] from segmentation-models library [269], with the ResNet-34 [270] backbone. For FCN and DeepLabV3 backbones we additionally use bottleneck blocks to increase speed [270] and dilated convolutions in the last 3 layers to keep the resolution reduction factor to 8 [267]. The dilation was necessary to obtain precise predictions.

For training we use AdamW [271] and binary cross-entropy (BCE) loss function. To satisfy memory constraints we use random crops of size 256×256, biased towards intensity values above 10. We find that further lowering of crop sizes decreases model performance. The use of intensity biased cropping in most of the cases reduced the training time and produced baseline models with metrics increased by few percentage points. Therefore, it was employed in all of the experiments. We train with batch size 16 for a maximum of 1000 epochs, selected from preliminary experiments on RealClutch. The initial learning rate and L2 weight decay factor were always selected using grid search over $\{10^{-3}, 10^{-4}\}$ and $\{10^{-4}, 10^{-5}\}$ respectively, maximizing F1 score on source validation set. The learning rate was halved every 50 epochs, with early stopping when relative decrease in validation loss is under 0.01 for 5 consecutive validations. Model parameters pre-trained on ImageNet did not improve speed or performance, similarly to [263]. Once the model is trained, fine-tuning is performed using real clutch images. For fine-tuning we only reduce the starting learning rate to 10^{-4} and train until convergence of validation loss. In all experiments, image values were centered to range $[-1, 1]$ and the defect mask channels were collapsed for single class segmentation. The reader is referred to the supplementary for detailed analysis.

The performance of our models and data sources is compared using pixel-wise metrics: precision, recall and F1 score. The model predictions were binarized with a threshold that maximizes the F1 score on validation set of the respective training dataset.

The RealClutch dataset consists of 3 correct and 3 defected objects. The objects were acquired using 86 viewpoints, covering all examined surfaces, resulting in 516 labeled images of resolution 2448×1025. The train-test split was constructed object-wise using 2 and 4 objects respectively, while keeping the 1:1 balance between the correct and defected samples. The train set contains objects with one surface texture, while the test set contains two surface textures which is common in evolving manufacturing processes. The train-val split was constructed with a 4:1 random split of the training set.

Source dataset	FCN			DLv3			U-Net		
	P [%]	R [%]	F1 [%]	P [%]	R [%]	F1 [%]	P [%]	R [%]	F1 [%]
RealClutch (baseline)	53.2	19.4	28.4	59.1	16.5	25.8	55.7	21.7	31.3
DAGM	0.0	4.6	0.1	1.0	0.1	0.1	0.1	5.9	0.2
KSDD2	1.5	5.1	2.3	2.2	6.5	3.3	0.7	7.0	1.3
Severstal Steel	1.0	9.7	1.8	1.7	9.0	2.9	1.0	3.2	1.5
MTD	7.7	10.2	8.8	23.5	9.9	13.9	8.7	11.6	10.0
CSEM-MISD	6.5	6.9	6.7	9.3	3.8	5.4	4.6	5.0	4.8
DAGM (FT)	15.5	4.7	7.2	9.3	5.3	6.7	20.0	3.3	5.6
KSDD2 (FT)	8.0	3.9	5.3	2.2	1.2	1.5	3.6	0.8	1.3
Severstal Steel (FT)	33.7	12.9	18.6	19.5	8.6	11.9	6.1	12.0	8.0
MTD (FT)	28.8	10.5	15.3	48.1	6.6	11.5	48.0	9.5	15.9
CSEM-MISD (FT)	55.2	17.5	26.5	55.0	19.9	29.2	46.3	13.5	20.9
SynthClutch	59.3	10.7	18.1	57.4	10.7	18.1	67.2	10.8	18.7
SynthClutch (FT)	**69.6**	24.0	**35.7**	63.1	25.5	36.3	**67.6**	**28.9**	**40.5**
RealClutch (EX)	59.0	16.3	25.5	54.8	17.9	27.0	55.2	20.3	29.7
SynthClutch (EX)	58.1	11.6	19.4	57.5	12.0	19.8	60.0	11.6	19.4
SynthClutch (EX+FT)	67.9	**24.2**	**35.7**	**67.6**	**27.7**	**39.3**	64.9	23.5	34.6

Table 7.1: Comparison between models trained on different source datasets, including fine-tuning (FT) and exposure stacking (EX), evaluated on RealClutch test split. Precision (P), recall (R) and F1 score (F1) are presented. The best results are bolded vertically.

The image resolution is halved for evaluation efficiency, padded to ensure divisibility by 96 and split into patches of size 416×352.

7.5.2 Effectiveness of Planar Datasets

When evaluating the usefulness of pre-training on existing datasets, the domain difference is compensated for using augmentations. We apply random rotations from $[-90, 90]$, Gaussian noise of variance $\leqslant 10$, exposures between $[0, 1]$, Gaussian blur with kernel sizes form $\{1, 3\}$, horizontal and vertical flips. For each dataset the model was trained on the source dataset and evaluated on the RealClutch test set with and without fine-tuning.

DAGM [258] is a synthetic dataset of generic textures with artifacts representing defects. It consists of 10 textures, totaling with around 8000 labeled samples for train and test splits. The labels are in form of ellipsoids surrounding the defected area which does not provide a detailed coverage of the defect and is a form of weak supervision. We pad the images to size 512 and split into patches of size 256×256.

Kolektor Surface Defect Detection v2 (KSDD2) [236] is a dataset for binary defect segmentation of metallic tile-shaped products with rough surface. The train split contains 2331 samples, which we split in 4:1 ratio for training and validation. The test split contains 1004 samples. For evaluation efficiency, we convert the images to grayscale, pad them with zeros to resolution 256×672 and split into patches of size 256×224.

Severstal Steel dataset [245] is a dataset of planar hot-rolled steel with

defects segmented in 4 classes which we collapse into binary segmentation. The test set for this dataset is private, however we use it solely for the pre-training of models so we utilize the available train split for training and validation. The train split contains 12568 samples, which we split using 4:1 ratio for training and validation respectively. We merge the defect classes into single class to make it compatible with our binary segmentation. For evaluation efficiency, we pad the images with zeros to resolution 1792×256 and split it into patches of size 224×256.

Magnetic Tile Defects (MTD) [249] is a dataset of magnetic tiles with slight curvature, used for saliency prediction over 5 defect classes. We select the defect classes that are important for our task based on their similarity to our data. We treat *blowhole*, *crack* and *break* as the defected class, while *uneven* and *free* are ignored. The *fray* class is not expected in our data and is thus ignored. Our subset of this dataset contains 1312 samples split into train-val sets using the 4:1 ratio, while keeping the ratio of correct and defected samples at 4:1 in both subsets. For evaluation efficiency, we standardize the resolution to 640×448 by padding with zeros and split the image into patches of size 320×224.

CSEM multi-illumination surface defects (CSEM-MISD) [237] collects 3 different objects (gear, screw and washer) used for defect segmentation. We use the highest 24 light points as defined in the paper, totaling to 2304 images. Different from the proposed method, we train the model to predict defects on each image separately. The train split contains 32 instances of every object which we split using 4:1 ratio for training and validation. For evaluation efficiency, we split the image into patches of size 256×256.

In table 7.1 generalization capabilities of models trained on different source datasets to the baseline model trained on RealClutch. As expected, the models trained on the RealClutch data generalize poorly due to the small number of available samples and overfitting due to training and validation being performed on different images but same objects. DAGM performs even worse since it does not model neither the surface texture nor the defect appearance of the metallic surfaces nor the tight segmentation masks. Severstal Steel shows some promise due to its size and defect variety which helps in regularizing the model to learn more robust features, which is especially visible after fine-tuning. MTD is most similar to surfaces of RealClutch and the results confirm this relative to other sources. CSEM-MISD additionally displays changes in defect appearance and after fine-tuning performs on par with RealClutch. Fine-tuning the models with RealClutch in most cases causes a significant increase of performance, with highest performance change attributed to the most similar datasets. However these gains cannot be predicted based on the pre-trained model performance. This allows the conclusion that the domain similarity in form of surface and defect appearance is important for knowledge transfer. Additionally, learning to correctly respond to different surface and defect appearances is a crucial information

for better performance.

7.5.3 Effectiveness of Custom Designed Synthetic Data

SynthClutch dataset consists of 20 correct and 20 defected object instances. We used the same viewpoints as for the real acquisition including the non-examined surfaces, totaling in 106 viewpoints and 4240 labeled images. The train-val-test split was constructed object-wise using $28-4-8$ objects respectively, while keeping the balance between correct and defected objects. We follow the augmentation from section 7.5.2, with max rotation angle reduced to 30 to avoid learning out-of-domain features.

Compared to the baseline model, the models trained on synthetic data produce lower recall and similar precision. However, fine-tuning on RealClutch boosts the performance above the baseline models by $5-10\%$ on both metrics. Most pronounced increase is in recall, where the model mostly increased the area of predictions to match the labels with a few additional defects becoming detected. When compared to pre-training on planar datasets, synthetic data doubles the model performance. This shows that task specific features guided by geometric attributes and surface texture are required for best prediction quality. Consequently, this simplifies the task for fine-tuning as model needs to adapt only to the smaller differences between domains.

The exposure stacking augmentation is evaluated only on SynthClutch since the defect behavior corresponds to the target RealClutch defects. As expected, in most cases recall increases as the model becomes more responsive to a larger surface area. However, the results are not consistently better or worse, indicating a need for more detailed research.

7.6 Discussion

Existing planar datasets were of limited value as sources of data for transferring knowledge to our geometrically complex domain. The real object contains sharp and curved geometrical features with tiny insignificant defects which can appear very bright under different views. Models tend to produce false-positives in those regions as they were not explicitly trained to ignore them. Even fine-tuning this does not fully resolve this issue, raising the importance of training on the target object data from the start.

The use of custom designed **synthetic data has proven to be the most promising approach when the amount of real data is extremely restricted**. Although still hindered by the domain gap, the overall model performance is significantly better than using models trained on alternative datasets, which in many cases contain more training samples but the domain is not similar enough. The model performance is additionally impeded by the task difficulty. Significant appearance changes of defects depending on the grazing angle and surrounding texture produce ambiguity

which would also be present for the human inspector. In such cases, a human inspector would not make a conclusion, but seek a different grazing angle, which was mimicked by the illumination stacking approach of Honzatko et al. [237]. In automated inspection the network is expected to decide based on a single view, which lowers the recall rate as observed in table 7.1. This comes from the fact that the synthetic data is overly precisely labeled - the defects are labeled if they are geometrically visible (not obstructed), and not if they are visible in terms of prominence. This is true for [237] as well. This issue hints at the need for models with efficient multi-view memory capabilities or models utilizing the grazing angle and location information about the view from a CAD model. So far, the research community has no answer to defect visibility evaluation, however our study raises this as an important point to be tackled in the future to prevent over-labeling in synthetic data.

The defect visibility problem is also closely related to the estimation of inspection coverage where we estimate if a region of the object surface can be inspected by a set of viewpoints. Our study on multiple exposures unveils the opportunity to study the *effective visual coverage* that is achieved by a particular recognition model, which greatly influences the process of inspection planning [8].

While fine-tuning on a small amount of real data helps with the domain gap, it is still prone to model overfitting due to high complexity of both the task and the model. [272] Further development of the data generator could reduce the gap from reality and reduce the need for fine-tuning. Enhancements might include texture models with richer variations or more precise selection of texture parameters. Both is achievable due to the immense controllability of generators based on computer graphics. Once designed, the data generator can reuse existing textures and defects, and be extended for the new ones. The extensions may be costly in terms of time however they become cheaper on the long run due to their high reusability and adaptability across different inspection targets.

7.7 Conclusion

When it comes to metal inspection, weighting the benefits of investing into a custom designed synthetic dataset against using publicly available datasets is difficult. Metal as a target domain is alone highly restrictive, whereas the multi-view inspection of complex metal geometry leaves us with a single publicly available real dataset. Therefore this work not only examined the benefits of the synthetic data, but additionally published a new dataset containing both real and corresponding synthetic data for multi-view inspection of a complex metal object. Such dataset is a first of its kind for metal inspection. The synthetic data has proven to be a superior pre-training data

source over multiple architectures but is still burdened by over-labeling. To resolve this, the research must further focus on generator enhancement, defect visibility quantification and utilization of object 3D as additional source of information for the network.

Core references

Juraj Fulir, Lovro Bosnar, Hans Hagen and Petra Gospodnetić, "Synthetic Data for Defect Segmentation on Complex Metal Surfaces", 2023. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4423-4433)., *doi:* 10.1109/CVPRW59228.2023.00465

Chapter 8

Image Synthesis Pipeline for Machine Vision in Metal Surface Inspection

Defect recognition is a crucial element of quality assurance in the manufacturing industry. With ever-improving industrial processes, defects become less frequent and harder to detect. Alongside, the variety of material surfaces is ever-increasing leading to difficulties in adaptation of surface inspection systems. All of this leads to slow iteration times of manually designed systems and not enough data for machine learning-based approaches. To solve this, we build on methods for image synthesis for surface inspection and parameterized texture and defects modeling introduced in Chapters 4, 5 and 6. We present a dataset consisting of synthetic and real data to demonstrate the utilization of Computer Graphics in data generation for the data-sparse task of defect detection.

For the sake of completeness, this chapter includes work done by PhD students Natascha Jeziorski and Juraj Fulir as well as work done by Tobias Herffurth. Tobias Herffurth performed test body design and material measurements (Sections 8.3 and 8.4). Natascha Jeziorski performed texture modeling based on the measurements (Section 8.5). Juraj Fulir performed acquisition and generation of dual dataset, quality estimation and pipeline evaluation (Sections 8.7, 8.8 and 8.9)

8.1 Introduction

Machine vision provides a way to automate repetitive processes which rely on visual information. As such, it has many uses in industrial applications. Those processes would otherwise either be performed by humans or could not be performed at all due to extreme working environment (temperature, noise, chemicals, etc.). As a technique, machine vision is very flexible since it

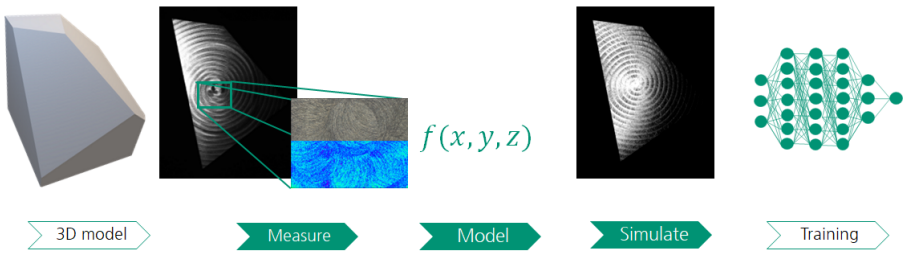


Figure 8.1: Presented synthesis pipeline. Based on a **3D model** of an object, spatial properties of the surface texture are **measured**. Surface topography and manufacturing parameters are used to develop a **mathematical model** capable of reproducing the texture as a normal map. Multiple realizations of the texture are generated by varying the parameters and are applied onto the 3D model of the object which has been altered to include surface defects of varying types and sizes. During the **simulation** step, the final image is computed based on the interaction between the acquisition environment (light, camera), object geometry and texture normal map. This process is repeated arbitrary amount of times, with different values and variance in order to generate a sufficient **training dataset**.

can be utilized for solving various challenges. However, every machine vision solution on its own is highly specialized, integrates expert knowledge and can not be easily adapted if inspection requirements or production environment changes.

Terms such as automation, flexibility, customization or smart systems are appearing more and more regularly in the context of manufacturing processes and production line optimization. This poses a particular challenge for automated visual inspection solutions since they are developed in a way which requires rigidness in order to ensure reliable detection precision [6]. In order to make the systems less rigid, the integrators of automated visual inspection systems are turning towards machine learning techniques for machine vision. While the proposed solutions sound promising, their success in terms of reliability and flexibility is yet to be proven, making the industry sceptical towards their integration into production lines. Additionally, machine learning requires a significant amount of training data, which has shown to be challenging in industrial environments. The tasks are typically very specific and there are little to no publicly available datasets. Therefore, a dataset must be created for each new system. This is not only costly and time-consuming, but also poses a significant challenge to generate a well balanced dataset.

Building on the inspection planning research done by Gospodnetic [3] and work introduced in Chapters 4, 5 and 6, we present a complete pipeline for generation of physically correct synthetic images, used for training sur-

face inspection machine learning models. As illustrated in fig. 8.1, the pipeline encompasses physical surface measurement, parameterization and development of stochastic geometry models representing the measurements, image synthesis, and dataset generation used for further training. The principles presented in this work can be applied to a large variety of materials and defects. For the purpose of this work, we focus on milled or sandblasted metal surfaces without coating, containing dent and scratch defects. While the measurement and modeling process can be performed for both texture and surface defects, for the purpose of this work, we will only present the texture modeling approach. Defect models will be used as defined in Chapter 6. The goal of our work is to perform industrial image synthesis for the controlled generation of data for surface inspection purposes. Synthetic image data generation provides control over content and diversity for dataset creation, speeding up the process and reducing its cost. As such, it has been used in many forms for machine vision such as optical flow, detection, segmentation, tracking, pose estimation, etc. as discussed by Nikolenko [214].

The pipeline introduced in Chapter 4 identified core computer graphics components needed to perform image synthesis for surface inspection and stressed the importance of texture and defect parameterization. The study presented a concept which opened a future possibility to ask and discuss questions arising from applying the pipeline to a realistic scenario. Therefore, our work extends the aforementioned pipeline and applies it to a realistic scenario. As such it introduces the following contributions:

- Extended data synthesis pipeline
- Introduction of surface and defect measurement procedure
- Separation of machined surfaces into micro, meso and macro geometry scales
- Introduction of requirements when developing stochastic geometry models which can faithfully represent the measured surface and defects
- Introduction of parameters as a point-of-control to synthetic dataset generation and variation
- Practical methodology for evaluating the quality of synthetic data
- Introduction of a public dual dataset for defect recognition over three texture groups, with a study of its quality

8.2 Synthetic Dataset Generation

8.2.1 Related Work

Synthetic image data generation can be roughly separated into **generative**, based on AI (predominantly deep learning) to produce synthetic data, and **rule-based**, using computer graphics simulation based on physically-based image synthesis, using a well-defined set of rules. The important difference between the two is that with the generative approach only a single realization is controlled while with the rule-based approach, the realization context is controlled, thus enabling complete control, reproducibility and reliability in terms of content.

Generative image synthesis approaches are often used when there is not enough time or knowledge available to create actual models simulating the process. Often, generative approaches are based on deep learning, specifically on generative adversarial networks (GANs) which are extensively discussed by Wu et al. [273], Gui et al. [274], Pan et al. [275], Figueira et al. [276] and Little et al. [277]. As such, GAN-based approaches were used in various computer vision tasks such as medical image synthesis [278], image super-resolution [279], image translation [280], texture synthesis [281], face synthesis [282], image inpainting [283], human synthesis [284], human pose synthesis [285], industrial quality inspection [286], [256], [229], [255]. Schmedemann [287] applied generative style-transfer on the synthetic image generated using a rule-based approach to reduce the *domain gap* - different between synthetic and real image. However, applying style transfer on images synthesized using the rule-based approach can lead to a loss of features making this approach not reliable without a human operator review of generated images.

Generative deep learning approach results in images which may look realistic but may also incorporate errors on a level which is beyond our comprehension, further causing misalignment when that data is used for training. Also, the generative methods do not know anything about the context or what has caused an image to appear in a particular manner, so it is not possible to control the dataset content and distribution of the features. That said, they can not introduce edge-case scenarios or guarantee correct data generation. This problem can be partially circumvented only by retraining on new data containing the desired features.

Rule-based image synthesis relies on computer graphics for modeling and rendering **virtual environments** which have recently gained popularity beyond the entertainment industry (e.g., movies or gaming). It represents a controllable, versatile and reliable approach for generating arbitrary amounts of data with customized features and variations. As such, it has proved useful across various machine vision tasks as discussed by Dahmen et al. [257]. There are several domains currently using synthetic data for ma-

chine vision tasks as surveyed by Tsirikoglou et al. [125]: human-oriented, autonomous driving and robotics. The surface inspection topic, covered in our work, can be considered to belong to industry automation, which is closely linked to robotics, but may include tasks beyond those needed for movement, localization and mapping. The domain of human-oriented synthetic data includes human body pose estimation [288], human body tracking [289], multi-person pose estimation and tracking [290], face recognition [291]. Therefore, the human-oriented domain focuses on large-scale geometry features leaving out realistic texture and detailed surfaces. The domain of autonomous driving relies on virtual environments of large-scale urban and traffic scenes [292] for recognition tasks [293] and object detection [294]. That said, virtual environments used for autonomous driving lack small-scale details and realistic textures. The robotics domain relies on virtual environments [295], [296] for recognition tasks such as semantic segmentation [297] and object detection [298]. Image synthesis for robotics aims for correct geometry representation while detailed surfaces are out of focus. The surface inspection domain relies on virtual environments for inspection planning and the development of defect recognition algorithms. The required level of realism and small-scale surface details for visual surface inspection make this domain highly different from other discussed domains. Virtual surface inspection planning was discussed by Gospodnetic et al. [188] and in Chapter 4. Procedural textures and defects for generating synthetic data for defect recognition in visual surface inspection were introduced in Chapters 5 and 6. A similar approach was taken by Schmedemann et al. [218]. A synthetic dataset for defect segmentation on metal surfaces was introduced in Chapter 7. Moonen et al. [223] presented a toolkit for synthetic image data generation in manufacturing. Synthesizing datasets for industrial inspection using a rule-based approach was further discussed by [299], [300], [219] and [260]. Synthetic data was further used for various quality assurance tasks such as inspection of industrial components [301], metal surface inspection [302], industrial visual inspection [303], scaffolding quality inspection [304], expected camera view [305]. General-purpose dataset generator is introduced by Greff et al. [306]. It promises realism (using physically-based ray tracing), scalability and reproducibility which are also covered by our pipeline. However, as it is general framework, it requires user to define a 3D scene for particular task using scripting. Assets for 3D scene must be either created by the user or imported from general-purpose asset libraries. As such, data generation for specialized problems, such as quality assurance, is challenging due to reasons such as unavailable assets, assets with unsatisfactory features or limited modeling capabilities of a user. In this work, we introduce rule-based image synthesis for visual surface inspection and with it, we introduce physically-based parameterization of simulated surfaces which existing pipelines do not provide. Our pipeline enables image synthesis controllability which goes beyond artistic

expression and relies on precise surface parameters.

In this work, **we introduce a rule-based image synthesis pipeline**, based on a virtual environment resembling the real inspection environment. Therefore, our focus lies in simulating small-scale surface texture and imperfections. Rule-based image synthesis pipeline for realistic image synthesis can be divided into two main steps: physically-based 3D scene modeling (i.e., modeling a virtual environment) and rendering procedures as discussed by Greenberg et al. [176] and Tsirikoglou [125].

3D scene contains 3D objects, lights and cameras. In this work, we focus on 3D object surface modeling, specifically surface texture and defects modeling. The realism of the surface highly depends on the texture which is used to describe surface topography resulting from machining. Surface topography highly influences surface reflectivity which is defined using bidirectional reflectance distribution function (BRDF) as discussed by Dorsey et al. [28]. Physically-based BRDFs are founded on microfacet theory [9] and describe local, small-scale light-surface interaction. Therefore, texture defines the variation of BRDF parameters over the surface as well as the variation of meso scale surface geometry on which BRDF is evaluated. Next to surface topography resulting from machining, we focus on surface defect modeling. Defect shapes are often complex resulting in complex light behaviour. To tackle this we build on the geometrical defect modeling pipeline introduced in Chapter 6. Although surface modeling is constantly moving towards physically-based methods respecting certain physical restrictions, such as conservation of energy (Heitz, [38], Hill et al. [307]), often the aim is visual appearance and support for artistic creation (Guerrero et al. [98], Adobe Substance [12], Ebert et al. [62]). Therefore, the development of models is not based on real-world parameters but rather artistic, making them visually appealing but not necessarily correct. In this work, we perform surface modeling using models based on real-world surface parameters as discussed in section 8.5.

Rendering is a process of generating 2D images from 3D scene and the crucial element for realism of generated images is light transport. In this work, we use state of the art, physically-based light transport based on path-tracing discussed by Kajiya [75] and Pharr et al. [14]. Therefore, in this work we use appleseed, a physically-based, path-tracing rendering engine [184].

8.2.2 Image Synthesis Pipeline

The image synthesis pipeline in this work is based on the description given in Chapter 4. For completeness, we provide an overview of the complete pipeline, extended with work introduced in Chapter 6. and our novel work regarding the usage of textures introduced in section 8.5 and required texture mapping. Image synthesis can be decomposed into 3D scene modeling and

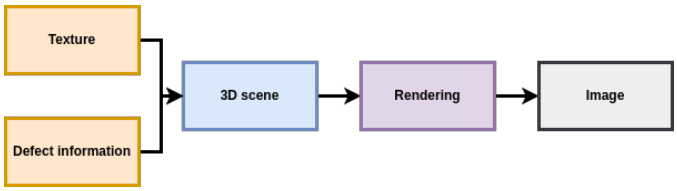


Figure 8.2: Image synthesis overview. In this work, we focus on 3D object surface and perform photo-realistic image synthesis of both defect-free and defected 3D objects with automated and pixel-precise defect annotations.

rendering which generates a 2D image from the 3D scene (see fig. 8.2). We first start by defining observation scales of 3D objects to provide the reader with the context on how the synthesis pipeline is structured, and what is important when it comes to surface inspection. Next, we provide an overview of 3D scene modeling introduced in Chapter 4, and extend it with texture mapping techniques which can handle complex 3D object geometry without requiring the UV unwrapping step (section 8.5). Texture mapping extension is needed since we are using image textures resulting from explicit procedural modeling and exemplar-based texture synthesis (section 8.5). We use those methods since they provide easier texture pattern modeling than implicit models. However, implicit procedural textures are far more suitable for automated generation since they can be evaluated on the fly during rendering as will be discussed in section 8.5. Further, we provide an overview of generating defect annotations, as introduced in Chapter 6, for defected geometry described in section 8.6.2. Finally, we provide an overview of the rendering, as discussed in Chapter 4, used in our work for photo-realistic synthesis of both defect-free object surfaces (fig. 8.5, fig. 8.6) and defected object surfaces with pixel-precise and automated annotations (fig. 8.7, fig. 8.9, fig. 8.10).

8.2.3 Decomposition of Scales

A 3D scene contains **3D objects** which are traditionally decoupled into geometry and material. The geometry specifies the size, position and shape of an object (high-level appearance), while the material influences how the light interacts with its surface and thus its detailed appearance. Alternatively, a 3D object can be considered at three different scales: macro, meso and micro. **Macro scale** refers to object geometry (e.g., mesh) and large geometrical features. **Meso scale** refers to the surface structure on a much smaller scale than the shape of the object, but larger than the wavelength of light (e.g., surface texture), as discussed by Dorsey et al. [28], ch. 2. **Micro scale** refers to surface structure and properties which are not separately distinguishable by the imaging sensor, but are contributing to light reflection.

It is important to note that it is very difficult to make a strict distinction

between the scales using units. This is because the scales are relative to the imaging sensor resolution and viewing distance. Therefore we propose the following guidelines. The macro scale should be attributed to geometrical shapes which take larger portions of the image. The micro scale can be determined using Nyquist-Shannon [308] sampling theorem stating that *the sampling rate must be at least twice the bandwidth of the signal.* Having defined what would constitute macro and micro scale, we can deduce that the meso scale constitutes features which occupy a smaller portion of the image, measured in a handful of pixels. With that in mind, we can also introduce a clear image scale separation between the meso and the micro scale: every feature which can not be sampled by more than one pixel falls under micro scale class.

For a more vivid explanation of the definition and scale relativism. In the case of visual surface inspection, we are looking at specific parts of an object with high resolution. Therefore the inspected object is in macro scale, its defects and surface texture caused by the manufacturing process are in the meso scale, while roughness and smallest geometrical features are all part of the micro scale. However, if we were to observe an outdoor scene of a park, containing many people and trees for example, the macro scale would be the shapes of people and trees, whereas the eyes or leaves would already fall under the meso scale, and the skin or the leaf veins would be considered micro scale.

8.2.4 3D Scene Modeling

The acquisition environment represents the context in rule-based image synthesis. As such, the scene to be simulated must faithfully represent the environment. For surface inspection environment this includes 3D objects, lights and cameras (see fig. 8.3). Notice that in this pipeline we do not model environment illumination, which is acceptable since surface inspection systems typically require a strictly controlled acquisition environment and no uncontrolled illumination.

In this work, following the scale decomposition introduced in section 8.2.3, the geometry of the inspected part is attributed to macro scale, texture and defects to meso scale and material reflectance and roughness properties to micro scale using BRDF. While all scales affect the realism, in this work we place particular focus on texture and defect modeling.

Geometry of 3D objects in 3D scene must be the same as real inspected products, which are in this work referred to as *test bodies.* Therefore, the 3D scene contains the same 3D geometry which is used for the production of the real test bodies (see fig. 8.11). Further, the geometry must contain defects which are as close as possible to real defects. Thus, product geometry is augmented using surface defecting models introduced in section 8.6.2.

3D object **material** is based on the state-of-the-art, physically-based,

microfacet-based **BRDF** for rough metal surfaces which is built on work by Walter et al. [9], Kulla et al. [309] and Turquin et al. [310].

The described BRDF is a parameterized function. The most important parameters for our work are BRDF roughness and surface normal in which BRDF is evaluated. Within this work, we refer to *surface normal* as *normal*. Using constant BRDF parameters and surface normals for evaluating the BRDF results in a smooth, overly perfect surface. Realistic surface modeling in computer graphics relies on varying BRDF parameters and surface normals using **texture**. This is highlighted in work by Dong et al. [174] where the texture was generated using surface measurements and Gaussian random field to add surface details on otherwise smooth and overly perfect metal surfaces. Although this approach gives impressive results, it does not provide the tools necessary to control the texture pattern based on real surface topography measurements. In our work, we build a completely controllable texture synthesis model using surface measurements.

Texture is a frequently used term in various domains. Therefore it can be ambiguous to the reader which may draw context from a domain other than computer graphics. In computer graphics, texture is a function which assigns values to each point of object surface to be used during rendering time. Texture describes surface patterns using vector values (e.g., normal) and scalar values (e.g., roughness) that are used during rendering which would require significant storage and computational efforts if they were to be represented by the actual geometry. Once synthesized, the surface will exhibit the pattern as defined by the texture (fig. 8.5, fig. 8.6).

In computer graphics, surface topographies can be represented using surface normals and BRDF roughness. In this work, we specifically focus on surface topologies resulting from machining discussed in section 8.3. Variation of surface normals and BRDF roughness over a surface is further represented using textures. That said, texture can be seen as a computer graphics tool to encode variations of surface topographies. Surface normal and BRDF roughness values generated using texture are used during rendering for evaluating BRDF model. Varying **roughness** parameter of BRDF over surface using texture results in variations in surface reflection [311]. Perturbing **surface normals** using texture, as discussed by Mikkelsen et al. [312], affects the orientation of BRDF, introducing the appearance of geometrical details (e.g., milling pattern) over a surface. In this work, we focus only on surface normal while leaving roughness to a small constant value (e.g., 0.0-0.1) due to the high level of detail modeled with surface normals.

Based on the topography measurements introduced in section 8.4, the two main surface characteristics are modeled, namely textures and defects. Texture modeling (section 8.5) focuses on developing algorithms capable of recreating sandblasted and milling surface patterns. The textures are generated as images to focus on physically based texture modeling without the additional layer of complexity introduced when adapting the models for an

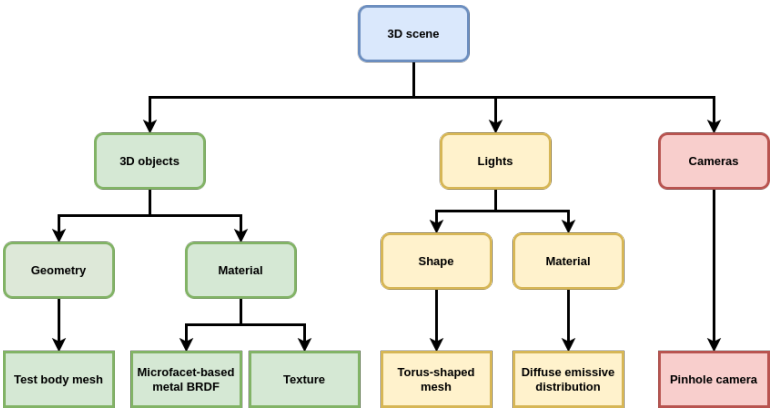


Figure 8.3: 3D scene decomposition.

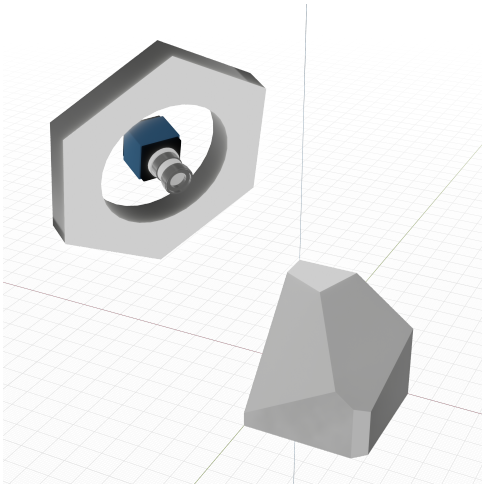
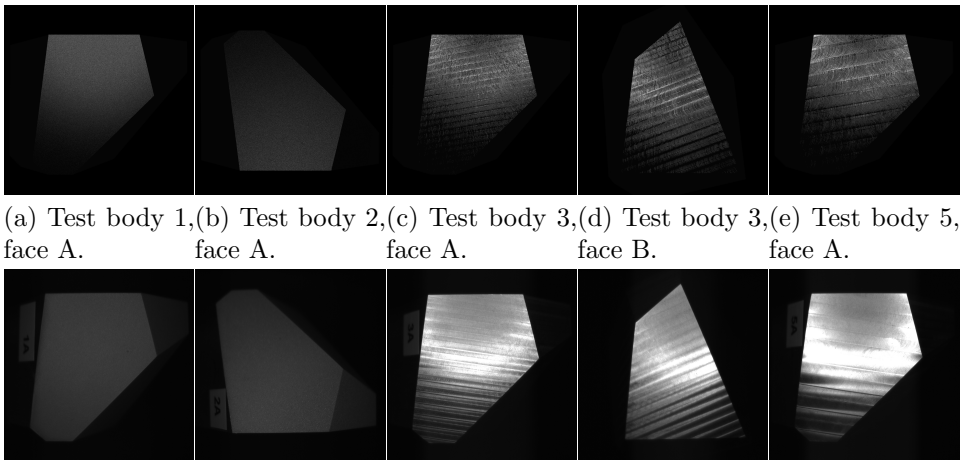


Figure 8.4: Simulated 3D scene representing the real inspection environment.

implicit procedural generation as suggested in Chapter 5. Sandblasted surfaces are modeled using a combination of exemplar-based methods whereas the model for milled surfaces is procedural but implemented explicitly for the sake of simplicity. Defect modeling (section 8.6) focuses on creating dent and scratch defects, resembling the ones present on the inspected product surface, as explained in section 8.3. Unlike the texture, which is modeled using normal mapping, the defects are modeled using actual geometry, which is embedded into the 3D object geometry. This is done because the defects are larger than the surface texture, and their complex shape causes intricate light scattering that can not be captured by normal maps alone.

Light in the 3D scene represents the illumination devices used in the surface inspection environment. Therefore, for the purpose of this work,



(a) Test body 1, face A. (b) Test body 2, face A. (c) Test body 3, face A. (d) Test body 3, face B. (e) Test body 5, face A.

(f) Test body 1, face A. Sanblasting 2.5 bar. (g) Test body 2, face A. Sandblasting 6 bar. (h) Test body 3, face A. Parallel milling 4mm, zustellfaktor 0.5. (i) Test body 3, face B. Parallel milling 4mm, zustellfaktor 0.8. (j) Test body 5, face A. Parallel 8mm, zustellfaktor 0.5.

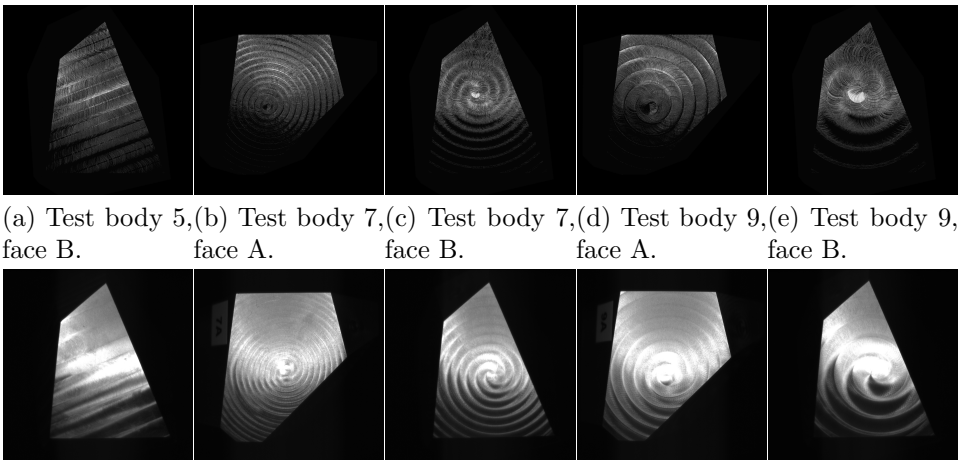
Figure 8.5: Top: synthesized images. Bottom: real images. The viewpoint is the 20-degree angle from a perpendicular view. Synthetic images contain slight variation since they are picked from the dataset.

the light geometry represents a model of the real ring light geometry used in the real inspection system which has a torus-like shape (fig. 8.4), with diffuse emission distribution describing the light emission in each point of the shape. This corresponds to the diffuse illumination device used during real acquisition. Light geometry is placed so that the camera is in the centre, with the same position and orientation as the camera, simulating the real setup where light is mounted on the camera.

Camera is based on a pinhole camera model defined by its resolution, pixel size, focal length, position and orientation parameters. Simulated camera parameters are set to resemble the real camera parameters used in the surface inspection environment. However, we do not account for aberrations such as distortions and depth of field of the real camera. Since the goal is to use synthetic data for ML where the focus is learning texture features and defects, camera aberrations can be introduced during training using augmentations. Therefore, the described camera model is enough for the pipeline we introduce.

8.2.5 Texture Mapping

The textures used in this work are created as normal map images, using explicit procedural models and exemplar-based texture synthesis (see sec-



(a) Test body 5, face B. (b) Test body 7, face A. (c) Test body 7, face B. (d) Test body 9, face A. (e) Test body 9, face B.

(f) Test body 5, face B. Parallel milling 8mm, zustellfaktor 0.8. (g) Test body 7, face A. Spiral milling 4mm, zustellfaktor 0.5. (h) Test body 7, face B. Spiral 4mm, zustellfaktor 0.8. (i) Test body 9, face A. Parallel milling 8mm, zustellfaktor 0.5 (j) Test body 9, face B. Parallel milling 8mm, zustellfaktor 0.8

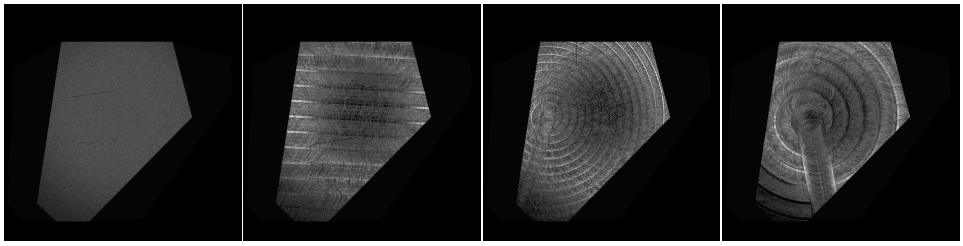
Figure 8.6: Top: synthetic images. Bottom: real images. The viewpoint is the 20-degree angle from a perpendicular view. Synthetic images contain slight variation since they are picked from the dataset.

tion 8.5), before the rendering takes place. Although implicit procedural models (Chapter 5) allow evaluation during rendering, and thus controllable on-the-fly texture generation are more suitable for automated generation, limited spatial information makes describing spatially-varying patterns with regular global structures, such as milling patterns present in physical test bodies (fig. 8.5, fig. 8.6, fig. 8.8), challenging.

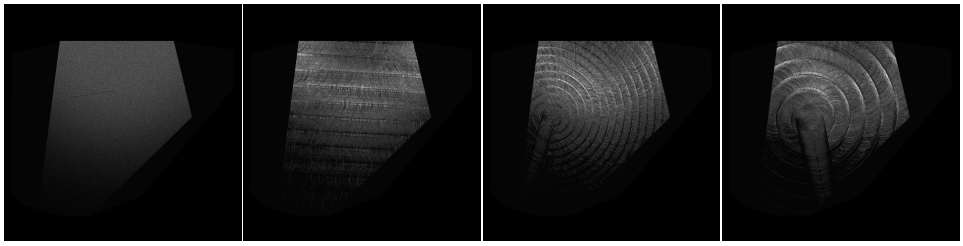
Normal maps are 2D image textures where each pixel encodes a normal vector. Such an array of normal vectors describes the meso scale surface pattern such as milling (fig. 8.6), present on the physical inspected object. To use created normal map images for evaluating BRDF during rendering, it is required to map procedural image texture on 3D object.

The way image texture is applied to an object greatly depends on the object's geometrical complexity. In the case of simple objects such as planes, cubes and spheres, texture can be easily applied using texture projection methods (e.g., planar, cube, spherical) as discussed by Pharr et al. [14]. In the case of more complex geometry (see fig. 8.11) those methods are not satisfactory if applied without care. For example, spherical projection does not fit the shape of object geometry (fig. 8.11) and thus causes texture stretching. Further, cube projection fails since object geometry faces (fig. 8.11) are under angle compared to cube projection planes which cause stretching. Therefore, more elaborate approaches must be used. One potential approach

is mesh unwrapping. Unfortunately, mesh unwrapping requires manual involvement and inspection of results which is typically performed by experts or 3D artists. Furthermore, physical samples created for this work contain different texture patterns per face. Setting up an unwrapped mesh to fit different image textures is not straightforward and is time-consuming. In our work, we take advantage of object shape and adaptively use texture projection methods. Since object geometry (fig. 8.11) contains planar faces, we use planar texture mapping *per face* and not on the whole object. First, 3D object geometry is separated on planar faces (fig. 8.11a). Second, planar texture projection is used on relevant faces. Third, mapped texture values are adapted to be used for BRDF evaluation during rendering.



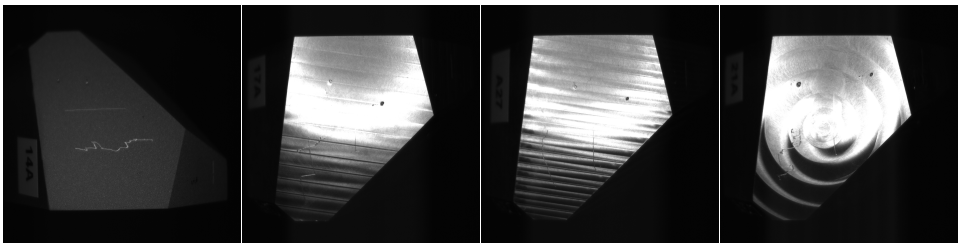
(a) Test body 1, face A. (b) Test body 5, face A. (c) Test body 7, face A. (d) Test body 9, face A.



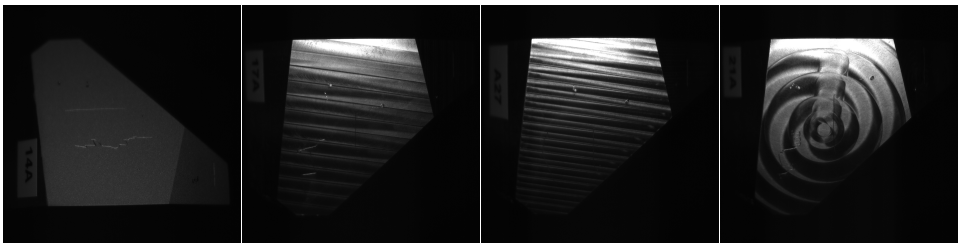
(e) Test body 1, face A. (f) Test body 5, face A. (g) Test body 7, face A. (h) Test body 9, face A.

Figure 8.7: Synthetic images of object surface with defects and textures. Top row: perpendicular view. Bottom row: 20 degrees angle from perpendicular view.

In the first step, 3D object geometry is separated into multiple faces to enable applying the texture on each face separately using the planar texture projection (fig. 8.11a). Pre-processing is performed on the original object geometry which is described parametrically (STP file format). Since each face is parametrically described, separation is performed using operators, such as *explode compound* for separating parametric geometry into faces, commonly provided in CAD modeling tools (e.g., FreeCAD [313]). Separated object geometry is grouped on relevant planar faces A, B, C and the rest of the object which contains non-relevant faces for our work (fig. 8.11a). Finally, tessellation is used on processed geometry to obtain a triangulated



(a) Test body 14, face A. (b) Test body 17, face A. (c) Test body 27, face A. (d) Test body 21, face A.



(e) Test body 14, face A. (f) Test body 17, face A. (g) Test body 27, face A. (h) Test body 21, face A.

Figure 8.8: Real images of object surface with defects. Top row: 20 degrees angle from perpendicular view. Bottom row: 40 degrees angle from perpendicular view.

mesh which is used for rendering.

In the second step, the separated object geometry is used in our 3D scene. Since each face is planar and has a certain orientation in 3D space, the planar texture projection [14] is used. Planar texture mapping only depends on the normal of the face onto which the texture is mapped. Since face normal is readily available and provided by the rendering method, texture mapping via planar texture projection can be performed efficiently.

In the third step, the processing of normals stored in the projected normal map is done before evaluating the BRDF. First, the projected texture is rotated and translated so that the position of the pattern matches the position of the pattern on the real surface. Next, red, green and blue (RGB) values are read from image textures. RGB values represent normal vectors and thus remapping from [0,1] (color values) to [-1,1] (normal vector values) must be performed. Further, normal vectors in image texture are stored in tangent space meaning that the Z axis is pointing up. Therefore, the face normal on which texture is mapped is used to construct orthonormal basis [314] which is then used for correctly orienting normal vectors from image texture. Finally, the normal obtained from normal image texture is used instead of the face normal and thus called perturbed normal.

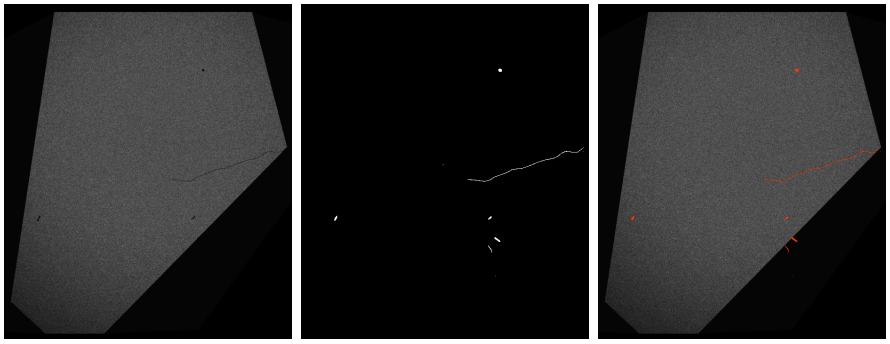


Figure 8.9: Left: defected synthetic image. Middle: defect annotations. Right: defected synthetic image with annotations overlay.

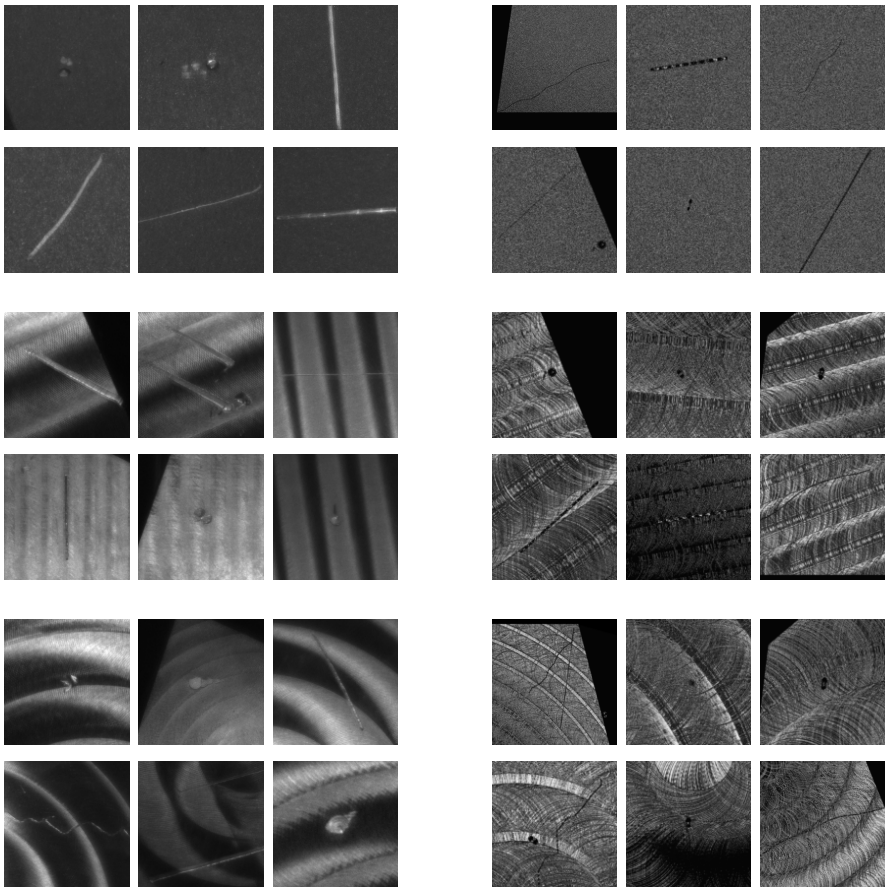


Figure 8.10: Real (left) and synthetic (right) crops of defects on different surfaces. Top: sandblasted. Middle: parallel milling. Bottom: spiral milling.

8.2.6 Defect Annotations Generation

Object geometry used in the 3D scene is defected using methods introduced in Chapter 6. Imprinting defects in actual geometry enables synthesizing images containing realistic defected surfaces (fig. 8.7, fig. 8.10). Alongside photo-realistic image synthesis of defected 3D objects, images containing defect annotations (fig. 8.9) are generated based on the description given in Chapter 6. For the sake of completeness, we provide an overview of the process of generating annotations since our goal is creating training-ready datasets which must contain precise annotations.

Images containing defect annotations are generated using both 3D object and geometrical defect masks (discussed in section 8.6.2). Each defect on 3D object geometry is covered by a geometrical defect mask. To distinguish defects, black (non-emissive) material is set to 3D object, emissive material is set to geometrical masks and all light sources are disabled. This way, only defect areas are visible from the camera position and not occluded by the 3D object are generated. Note that this approach also generates defecting annotations on any part of the object visible from the camera, even if the object is in the dark. Finally, generated images containing defect annotations are paired with photo-realistic images using the same camera parameters, effectively achieving automatic and pixel-precise defect annotations.

8.2.7 Rendering

Once a 3D scene is modeled, rendering is performed to create a 2D image from a particular camera position using a 3D scene description. The main rendering parameters for the quality of synthesized images are the number of samples per pixel (SPP) and the number of light ray bounces.

Each pixel in the 2D image, depending on the camera position, covers a certain portion of the 3D scene. In our work, pixels cover rich and complex surface structures. Therefore, we use a higher number of SPP (i.e., 128) to ensure the computation of pixel color which approximates well the surface covered by that pixel. A higher number of SPPs effectively reduces noise and aliasing in the rendered image.

In our work, 3D objects contain complex, geometrically imprinted defects. To simulate correct light transport and obtain realistic defect response to light, multiple light ray bounces are used. This way, complex light phenomena such as shadowing, masking or interreflections expected to happen on complex geometry is well approximated (fig. 8.7).

8.3 Test Body Design

The image synthesis pipeline presented in this work builds upon manufacturing domain knowledge to reproduce photorealistic data. Therefore, to

remove any potential ambiguity coming from manufacturing, we use custom designed and manufactured test objects. In return, this provides us with complete insight into and control over the processing chain, allowing us to focus on the image synthesis, surface modeling, dataset generation and training challenges.

In total, three identical sets of test objects were fabricated, each containing 10 test bodies. The bodies within the set vary in geometry and surface machining methods. When designing the test bodies, the aim was to generate bodies with multiple test faces (surfaces) showing different realistic surface texture patterns. The base bodies (type B and type M) are polygons made from aluminum, see fig. 8.11a. Each polygon has size 5cm×5cm×5cm with planar faces A, B and C of approximately size 2cm×2cm which are measured and modeled subsequently. The remaining faces are left "unfinished" (rough milling) and are not further considered. The difference between type B and type M is the mirrored arrangement of plane faces of the base body.

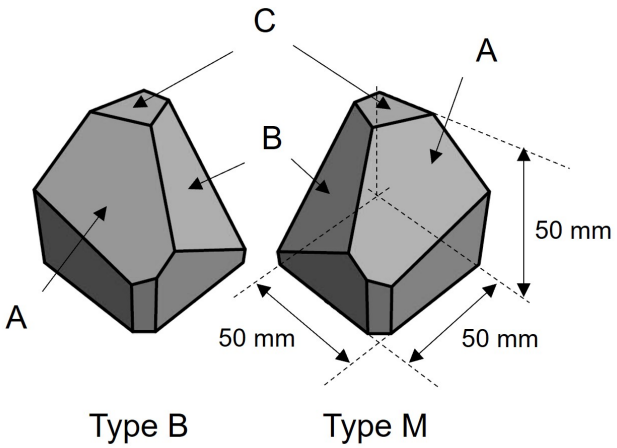
The scope of this work is restricted to milling and sandblasting surface processing methods.

First, the surfaces are milled using different processing parameters (fig. 8.11b). This is a common machining method for the shaping of work pieces. It uses a rotating cutter with a specific head diameter to remove material from the surfaces by performing many separate, small cuts along the tool path with a defined feed rate. Here, we consider face-milling, which means that the tool moves perpendicularly to the object surface. The distance between the neighboring tool paths depends on the radial depth of the cut and influences the resulting tool marks, see fig. 8.11c. Parallel and spiral tool paths are used, i.e. the surface is milled in parallel lines or in a spiral way. Artifacts in the form of exit lines can occur for spiral milling (fig. 8.8 and fig. 8.7).

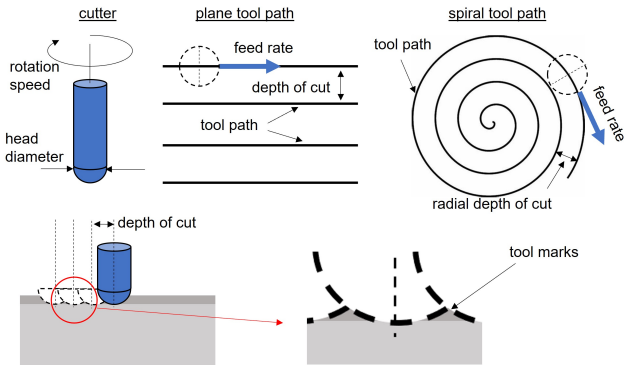
After milling, some of the faces were sandblasted, i.e. a hand-held nozzle of a sandblasting tool blasts the surface with a mixture of sand and air under high pressure. This abrasive method can smoothen or roughen the surface, dependent on the pressure of the air jet. The resulting surface topography dependents thus on the pressure of the air jet and the grain size distribution, shape and material of the sand.

The parameter values were chosen to create realistic patterns that resemble the industrial processing of products. For milling, this includes the selection and variation of the milling head size, the feed rate or the radial depth of cut and for sandblasting the variation of the pressure. All the applied process parameter settings are summarized in section 8.3.

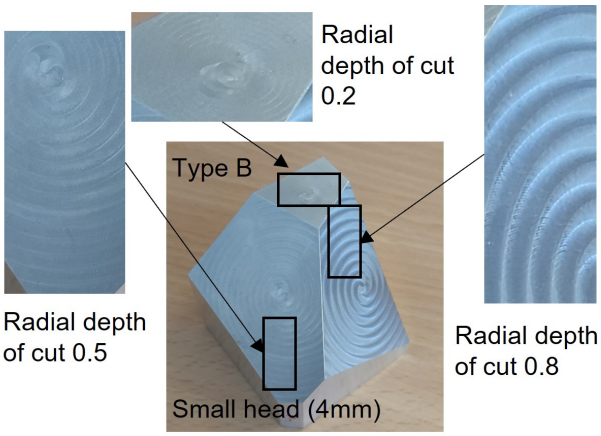
Objects in set 1 contain no surface defects, while sets 2 and 3 contain more than 200 scratches and localized point defects, created precisely on the faces. For this purpose, a custom "indenter" and a scratch test tool were used where the tips can be loaded with different masses. As tips we used a diamond tip, a steel needle and a screw. The load of the tips were 500g, 1000g and 1500g. Using the tools, the scratches (straight and free-form



(a) Drawing of the base test bodies.



(b) Sketch of the milling processes and its main parameters.



(c) Picture of the manufactured surfaces of type B spiral milled with a small head (4mm) and various radial cutting depths.

Figure 8.11: Illustration of the test object used in the project, the milling processes and a test object with milled surfaces.

manufacturing technique	parameter	values
milling	milling head diameter	4mm, 8mm
	radial depth of cut	0.2, 0.5, 0.8
	path	parallel, spiral
sandblasting	pressure	2.5bar, 6bar

Table 8.1: Used parameter settings for processing.

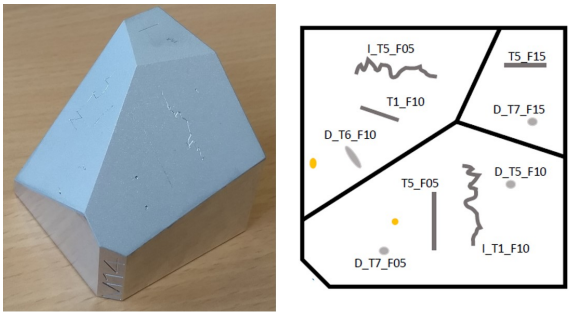
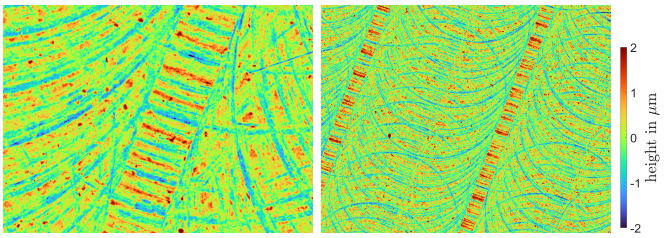


Figure 8.12: Test object type M after sandblasting and subsequently introduced defects with different types and sizes.

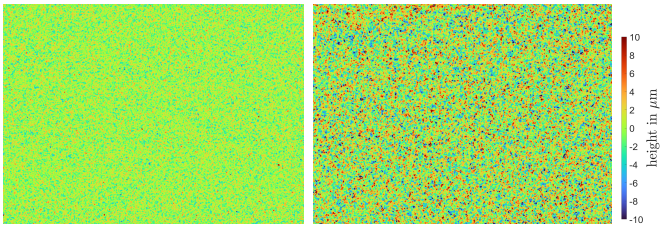
lines) and digs (dents) are realized on the surfaces as typical surface defects in accordance to ISO8785. The generated defect sizes were in the range of sub-millimeter to millimeter. See fig. 8.11 for an example.

8.4 Material Measurements

The focus variation microscopy is a non-contact high-resolution surface measurement technique, but can also be used for lower-resolution surface measurements of larger surface areas. In order to characterize the machined surfaces, where both the small details and the surface pattern are important, optical 3D and topography measurements are performed by means of focus-variation microscope (Bruker alicona InfiniteFocus G4). In the high precision machining industry measurement systems based on the focus-variation technique are standard inspection tools to characterize 3D mechanical parts. The advantage is the high precision and contact-free measurement of roughness, surface structure, micro-geometry and form using one optical sensor. The measurement system is based on a precise optical lens system with shallow depth sharpness. By changing the working distance between the measured surface and the microscope lens, different depths of the surface come into focus and are projected sharply on to the sensor. The proprietary software analyzes the distance for each point and measures the sharpness,



(a) Parallel milled surface using milling head diameter 4mm and depth of cut 0.5. Small imaged region (left) is 2mm × 1.4mm using $\nu_{\mathcal{M}} \approx 0.44\mu$m and large imaged region (right) is 7.5mm × 5.9mm using $\nu_{\mathcal{M}} \approx 1.75\mu$m.



(b) Sandblasted surface using pressure 2.5bar (left) and 6bar (right). Imaged region is 7.7mm × 5.7mm using $\nu_{\mathcal{M}} \approx 1.75\mu$m.

Figure 8.13: Topography measurements of sandblasted and milled surfaces.

which is used for the calculation of the surface depth profile (topography). Depending on the magnification of the optical lens system, this process allows a vertical resolution down to 10nm (magnification equivalent of 100x, which is a physical magnification limit). The measurements are provided as 2D images $\mathcal{M} : \{1, \ldots, M_{\mathcal{M}}\} \times \{1, \ldots, N_{\mathcal{M}}\} \to \mathbb{R}^d$ of size $M_{\mathcal{M}} \times N_{\mathcal{M}}$ for $M_{\mathcal{M}}, N_{\mathcal{M}} \in \mathbb{N}$ and pixel spacing $\nu_{\mathcal{M}} \in \mathbb{R}_{>0}$. The topography images can be interpreted as height images, that means, each pixel is assigned a height value.

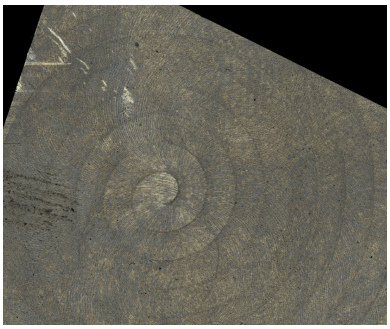
For the purpose of this work, an optical lens system with a magnification of 5x and a nominal vertical resolution of 410nm is used. The vertical resolution was limited manually during data acquisition by the measurement software. The defect free samples of objects in test set 1 are measured with different lateral resolutions, depending on the size of the scanned surface area, as it is shown in fig. 8.13. The topography measurements are converted into ASCII xyz-files in order to create readable files used for surface modeling.

Depending on the process parameters, the tool marks created by the milling process form an unique pattern which can be observed by the naked eye (fig. 8.11c) and influences the subsequent surface light response, playing the key role in surface inspection. Measurements of surfaces processed using different parameter settings are given in fig. 8.14. When describing the surface topography it has to be distinguished between surface roughness

Parallel, milling head diameter 4mm, depth of cut 0.8.

Parallel, milling head diameter 8mm, depth of cut 0.8.

Spiral, milling head diameter 4mm, depth of cut 0.5.



Spiral, milling head diameter 8mm, depth of cut 0.2.

Figure 8.14: Optical 3D measurements of milled surfaces using different parameter settings. Imaged region is 21mm × 17.5mm using $\nu_{\mathcal{M}} \approx 7\mu$m.

and waviness. The latter is more on the long scale range and has typically a periodical structure (e.g. tool marks). On smaller length scale, the tool marks are overlayed with stochastically distributed fluctuations of the surface height (spacing of the irregularities $\ll$ wavelength of waviness) which is described by the surface roughness. For example, the surface roughness can be influenced by the microscopic structure of the work piece, e.g. material. The measured heights of the waviness are in the range of several micrometers whereas the resulting root mean square (rms) roughness is about 2μm. When sandblasting, the previous milling does not affect the final surface pattern, see fig. 8.13b. It results in a random stationary and isotropic homogeneous surface without spatial long scale relations.

In order to characterize the generated defects we used an optical lens system with a magnification of 20x and adjusted the vertical resolution during data acquisition in the software manually (depth quality filter value between $2e-6$ and $8e-6$ depending on the defect type and size). See fig. 8.15 for an example. The shape of the dent resembles a dig with a peak and a valley of about 150μm in height and depth with respect to the undamaged surface

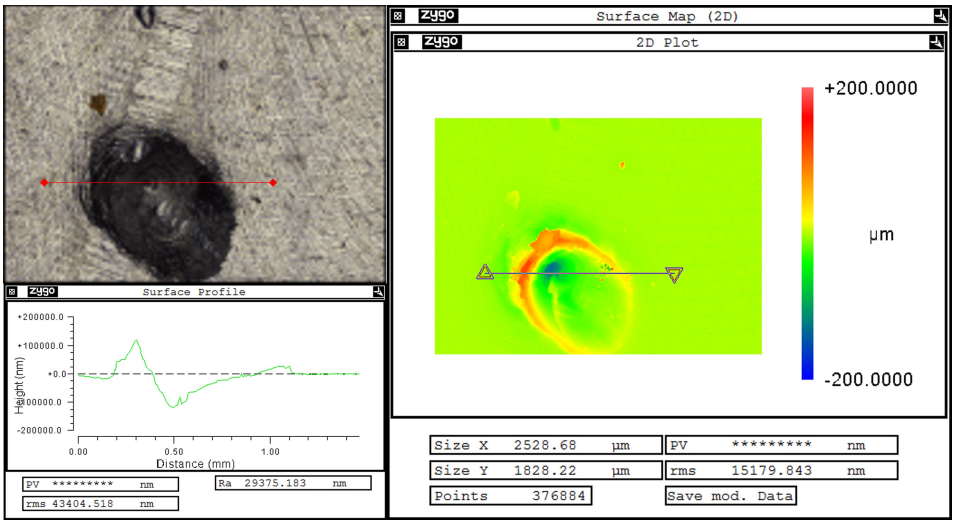


Figure 8.15: Optical 3D (top left) and topography (right) measurement with 2D intersection of the height profile (bottom left) marked in the topography measurement of a defect created using the indentor with load of 500g.

level.

8.5 Texture Modeling

The sandblasting and milling surface processing methods described in section 8.3 resulted in very different surface topographies (fig. 8.13). Therefore, separate models are required for both processing methods, each based on the corresponding topography measurements $\mathcal{M}$. The surface topography describes the shape of an object in meso scale as introduced in section 8.2 which is here given as texture. For the sake of simplicity, texture images are used. Thus, the output of the models is a texture image $\mathcal{T}$ of arbitrary size $M_\mathcal{T} \times N_\mathcal{T}$ and pixel spacing $\nu_\mathcal{T} \geqslant \nu_\mathcal{M}$ coarser or equal to the input's pixel spacing. For the purpose of this work, we use $\nu_\mathcal{T} \approx 6.1\mu$m and $M_\mathcal{T} = N_\mathcal{T} = 13107$ so that a squared imaged region with edge length 80mm is provided. The chosen pixel spacing is large enough in order to generate images of appropriate size for which the rendering process can still be done in acceptable time. However, pixel spacing is chosen small enough so that no visible discretization artifacts occur. The resulting texture image can completely cover test body surfaces during the later rendering process, without the need for image tiling, even when the texture image is rotated. Tiling is avoided because it repeats the texture image if it is too small to cover the considered surface, producing visible edges between the tiles. Such edges break the texture pattern and are unrealistic.

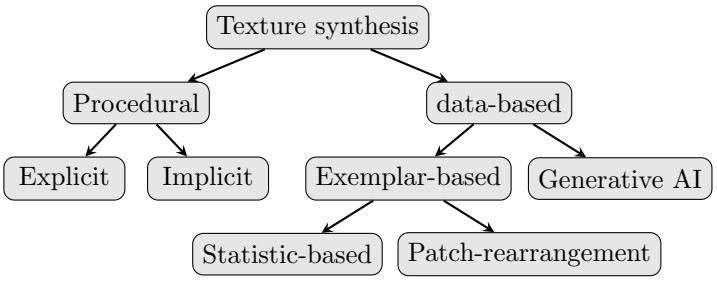


Figure 8.16: Classification of computer graphics texture synthesis methods.

8.5.1 Related Work

Particularly interesting computer graphics methods for modeling highly reflective metal surfaces are the ones describing highly specular surfaces with high-frequency details called glints discussed by Zhu et al. [315] and Chermain [316]. Glint effects appear due to complex and unstructured small-scale, high-frequency geometries such as tiny bumps, dents and scratches. The problem with these approaches is that they do not model the actual properties of surfaces to resemble any particular and standardized surface. Rather, they are modeled to resemble the surface finishing appearance in an artistic way which is difficult to compare to any particular real-world counterpart.

To our knowledge, there is no work describing texture synthesis models for sandblasted surfaces. Milling belongs to the machining processes which are deterministic and standardized, and therefore result in deterministic and describable surface patterns as discussed by Groover et al. [317] and Childs et al. [318]. Therefore it is extremely useful to incorporate existing domain knowledge while modeling textures for milled surfaces. In order to estimate the surface quality of milled surfaces and thus determine the required quality Felho et al. [319,320] and Kundrak et al. [321] developed models for parallel face-milled surfaces. Further, Hadad et al. [322] developed a more detailed model that takes other typical milling parameters into account, e.g. tilting of the tool to avoid re-cutting. All models create milled surfaces using the CAD software and the patterns are thus geometrically imprinted into the surface. In our application, that procedure would lead to extensively long computation times for the rendering step. Furthermore, the surfaces generated in this way look too perfect, as possible irregularities are not included in the models, e.g. slight deviations in the cutting geometry due to irregular material and machine behavior.

In computer graphics, surface topographies can be represented by textures which are roughly separated into procedural and data-based. The whole classification tree is illustrated in fig. 8.16.

Procedural texture modeling is based on algorithms which com-

pletely describe the surface patterns without the need for additional data (e.g., image data). The main concept is to combine regular patterns (e.g. sine, simple shapes such as lines and circles as well as other mathematical functions) discussed by Vivo et al. [63] with irregular patterns (e.g. noise functions) discussed by Dong et al. [53]. Since procedural texture models are algorithmic descriptions of a texture, they offer parameterization and thus immense controllability which is recognized by the computer graphics community for content creation [10]. However, developing procedural textures is often done by artists who base their work on experience and expression rather than encoding physically correct parameters.

Deguy [10] separates procedural models into implicit and explicit. The **explicit** procedural texture models generate texture images which are first mapped on the object surface and then used during rendering. Contrary, the **implicit** procedural texture models are directly evaluated during the rendering procedure ensuring texture generation in render-time and allowing render-time re-parametrization. Moreover, implicit procedural texture models can cover arbitrary large surfaces without repetitions or seams and they provide infinite resolution. On the other hand, explicit procedural models are easier to develop for complex texture patterns. In Chapter 5 implicit procedural texture synthesis method for circular, parallel and radial textures was presented. Those patterns resemble processed surfaces but no explicit machining method is taken into account.

Data-based texture modeling relies on sensor data and measurements as input, as discussed by Tsirikoglou et al. [125]. For example, the input texture image may be obtained using photogrammetry [51] or other scanning techniques [323]. The advantage of data-driven texture modeling is to synthesize approximations of real-world surfaces for which there are no well-defined mathematical descriptions. However, those methods offer little control over the generated content since they depend entirely on the input data. Thus, no structure-describing input parameters exist. Data-based texture modeling can be roughly separated into generative AI and exemplar-based. **Generative AI texture synthesis** is often based on GANs as discussed by Jetchev et al. [324], Bergmann et al. [281] and Zeltner et al. [325]. On the other hand, **exemplar-based texture synthesis** uses algorithms to generate new texture images similar to a given input image. While stationary textures can be well reproduced, problems occur for globally varying textures. Further, since the output image relies entirely on the input image, it leaves no possibility of influencing the pattern itself. Raad et al. [326] give an overview of common exemplar-based texture synthesis methods. They distinguish between statistic-based and patch-rearrangement methods.

Statistic-based exemplar-based methods divide into two steps. First, model-dependent statistics of the input image are computed followed by the adjustment of a random image so that its statistics match. The asymptotic discrete spot noise (ADSN) and the random phase noise (RPN) are both

parameter-free and non-iterative statistic-based methods [327]. A texture image simulated by the ADSN maintains the mean of the input image and its auto-covariance equals the input's auto-correlation. The RPN creates texture images that have the same Fourier modulus as the input image but a random Fourier phase. Heeger and Bergen [328,329] introduced a different procedure for texture modeling that uses image pyramids. A random image is adjusted iteratively by applying histogram matching to all images in its image pyramid according to the image pyramid of the reference image. Portilla and Simoncelli [330,331] improved this method by using certain statistics instead of histogram matching, e.g. cross-correlations between pyramid levels. In contrast to the exemplar-based methods that generate completely new texture images, the **patch-rearrangement methods** quilt patches taken from the input image. With the method of Efros and Freeman [332, 333], the output image grows successively by adding certain patches one after the other in a raster-scan order. It is an extension of the method of Efros and Leung [334] that generates a new texture image pixel by pixel.

Guehl et al. [335] introduce a hybrid approach combining procedural and data-based texture synthesis. The visual structure of the generated image texture is based on a procedural parametric model, while texture details are synthesized using a data-driven approach. Another hybrid approach is presented by Hu et al. [113], in which the texture of the input image, which contains the BRDF parameters, is decomposed and proceduralized.

8.5.2 Sandblasted Surface

When sandblasting, the surface topography changes with respect to the range of height values and the degree of roughness depending on the pressure of the air-jet (fig. 8.13b). The resulting surface topography is homogeneous, resembling a stationary Gaussian random field. However, fitting Gaussian random fields did not provide satisfactory texture images. In addition, due to the small number of process parameters, it is difficult to develop a model that depends exclusively on them. Therefore, we used a combination of exemplar-based texture synthesis methods that only receive the measurement as input. We select measurements of approximately size 4440×3250 and pixel spacing $\nu\mathcal{M} \approx 1.75\mu$m that correspond to an area of size 7.5mm $\times$ 5.7mm.

To generate texture images for sandblasted surfaces, the asymptotic discrete spot noise (ADSN) is used, introduced by Galerne at al. [327]. The size and pixel spacing of the input image is maintained. To obtain the new texture image, a convolution between a Gaussian white noise image and the input is used. Thus, the output is normally distributed having as mean and auto-covariance the input's sample mean and auto-correlation. The measurements serve as input for the ADSN.

Next, the desired size $M_\mathcal{T} \times N_\mathcal{T}$ and pixel spacing $\nu_\mathcal{T}$ of the texture

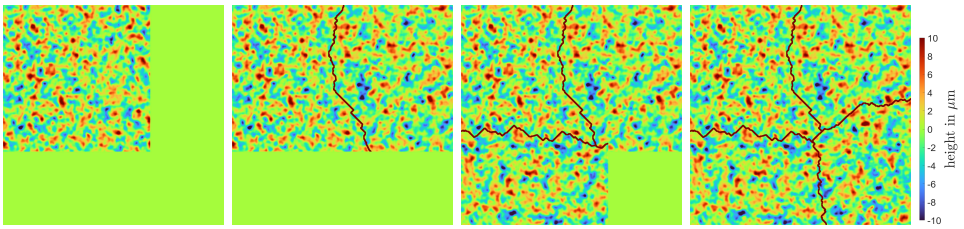


Figure 8.17: Illustration of EF-stitching marking the minimal path in each step. Patches have the size of 512×512 and the overlap region width is 256 pixels. The imaged region is approximately 1.34mm $\times$ 1.34mm.

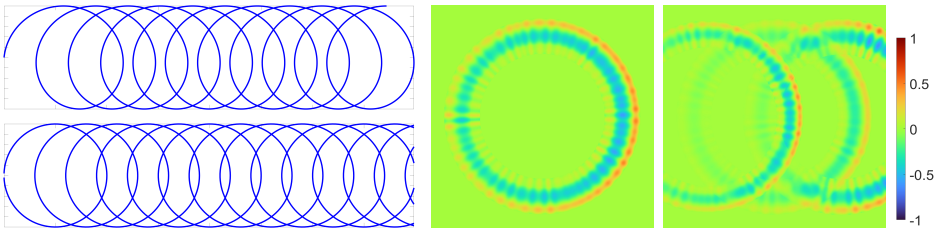
image have to be met. The ADSN can generate texture images larger than the input image, while maintaining the same pixel spacing. Therefore, the input image is enlarged using padding with its mean value before applying the ADSN. However, periodical repetitions are visible. To avoid that circumstance, multiple texture image patches are generated and stitched using EF-stitching. The patchwork texture is then down-sampled using nearest neighbor interpolation to meet the desired pixel spacing.

EF-stitching was introduced by Efros and Freeman [332, 333]. To stitch two neighboring patches within an overlap region, both are cut apart along an intersection edge first and put together afterwards. The intersection edge is determined by finding the path within the overlap region where both images are most similar. Therefore, the minimal path approach is used. Depending on the alignment of two neighboring patches we distinguish between vertical, horizontal and L-shaped stitching (fig. 8.17).

8.5.3 Milled Surface

The model generating textures of milled surfaces is given as a function in a continuous domain $\mathbb{R}^2$. Thus, it is procedural but implemented explicitly for the sake of simplicity. The discrete texture image is obtained by evaluating the function at the image grid-points $(\nu_{\mathcal{T}} x, \nu_{\mathcal{T}} y)$ for $x = 1, \ldots, M_{\mathcal{T}}$ and $y = 1, \ldots, N_{\mathcal{T}}$. The milling pattern appearance depends on a number of parameters associated with production parameters. The most prominent structures are cycloidal edge paths caused by the rotation motion of the milling head. In this work they are approximated by $n \in \mathbb{N}$ rings $R_1, \ldots, R_n$ (fig. 8.18a). The model is further divided into the following sub-models for the

1. tool path providing the arrangement and order of the rings by defining their center points c_k
2. appearance of an individual ring R_k (fig. 8.18b)
3. interaction between neighboring rings (fig. 8.18b)



(a) Edge path (top) and its approximation (bottom).

(b) Sub-models for one ring (left) and interaction of multiple rings (right).

Figure 8.18: Overview of model generating milled surfaces.

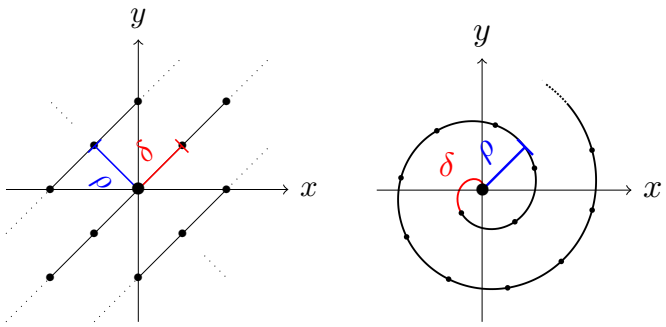


Figure 8.19: Tool paths of parallel (left) and spiral milling (right) with center points.

The first sub-model generates the ring center points $c_k \in \mathbb{R}^2$ along the tool path. Within this study, either parallel or spiral milling (fig. 8.19) is considered. First, the respective tool path has to be modeled, together with the distance $\rho \in (0, d)$ between two neighboring sub paths. The distance is controlled by the amount of ring overlap $\alpha \in (0, 1)$ and the diameter $d \in \mathbb{R}_{>0}$ of the rings, as $\rho = (1 - \alpha) \cdot d$. The diameter of the milling head is crucial for defining the ring diameter and the radial cutting depths for the ring overlap $\alpha = (1 - \text{cutting depth})$. As the blades do not reach the edge of the tool, α is reduced here by taking $\alpha - \gamma$ with $\gamma \in [0, \alpha)$. Then, points $\tilde{c}_k$ are computed that are located on the tool path to get the ring center points by $c_k \sim \mathcal{N}(\tilde{c}_k, \Sigma_c)$. Their location is specified by the distance $\delta \in \mathbb{R}_{>0}$ between two center points along the tool path. The distance δ depends on the tool's feed rate and rotational speed. Further, it is important to keep the chronological order of the rings in order to keep the tool path visible. Finally, while being very precise, the milling process is not perfect and every once in a while there is an occurrence of a ring which is more visible than the others. To introduce such irregularities in visibility of rings, the order of a certain amount $\epsilon \in [0, 1]$ or rings is changed.

The second sub-model controls the appearance of an individual ring R_k,

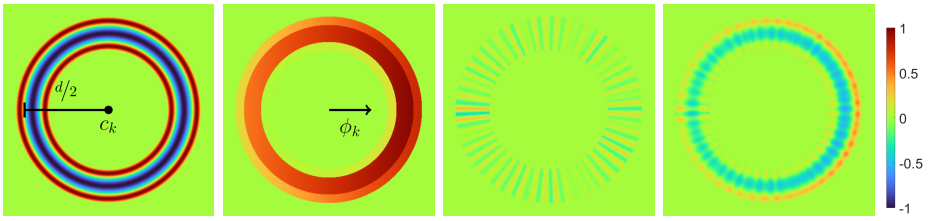
which is the result of the cut within a single rotation of the milling head, produced by a cutting edge. Therefore, the width of the cutting edge is crucial for the indentation width $w_k^- \sim \mathcal{N}(\mu_{w^-}, \sigma_{w^-})$. To get height values, the shape of the indentation is modeled by the cosine function $-\cos(x)$ for $x \in [-\pi/2, \pi/2]$ scaled to the width of the ring. Material depositions are modeled by accumulations beside the indentation in form of rings having positive values. The width for the inner and outer rings are $w_k^{+i} \sim \mathcal{N}(\mu_{w^{+i}}, \sigma_{w^{+i}})$ and $w_k^{+o} \sim \mathcal{N}(\mu_{w^{+o}}, \sigma_{w^{+o}})$ and the height component is again modeled using the cosine function. All together is called the shape S_k of ring R_k (fig. 8.20).

So far, perpendicularity between the milling head and the surface has been assumed. However, the milling head is often tilted forwards in direction of the tool movement to prevent re-cutting the surface that was treated just shortly before. It is given by the angle $\phi_k \in (-\pi, \pi]$ in the x-y–-plane between the tool path and the x-axis at point $\tilde{c}_k$. Tilting introduces a slope within the rings defined by the minimal $l_k^\bullet \sim \mathcal{N}(\mu_{l^\bullet}, \sigma_{l^\bullet})$ and maximal $h_k^\bullet \sim \mathcal{N}(\mu_{h^\bullet}, \sigma_{h^\bullet})$ indentation, inner and outer accumulation at the ring's front respectively back for $\bullet \in \{-, +i, +o\}$ (fig. 8.20). Since the tool's presence prevents pushing material to the inside, l^{+i} and h^{+i} are chosen small. Tilting T_k is then applied by multiplying with the shape S_k.

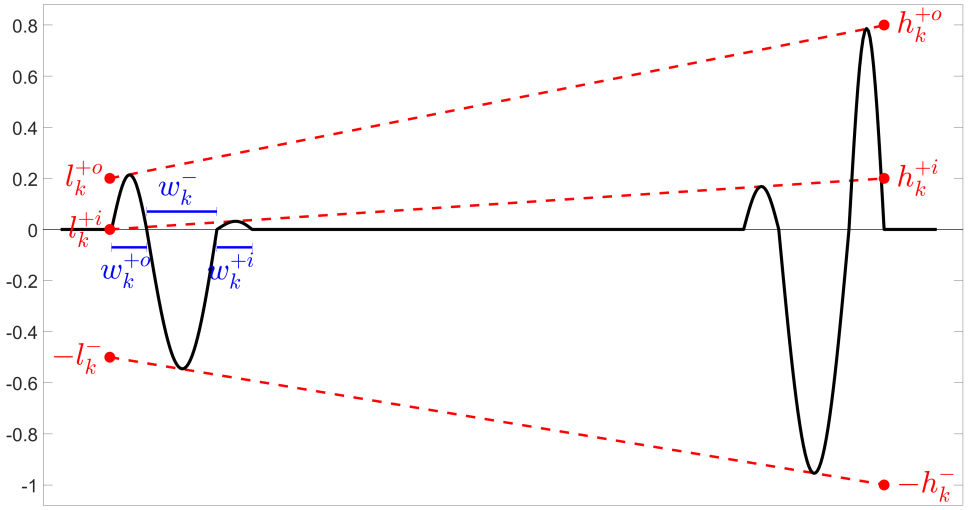
Finally, noise N_k is added, simulating irregularities caused by resistances in the material or vibrations during the process. Therefore, a combination of $\lambda_k \sim \mathcal{P}(\lambda)$ sine curves with varying frequencies $\tau_{k_j} \sim \mathcal{P}(\tau)$ and random shifts $\xi_{k_j} \sim \mathcal{U}((-\pi, \pi])$ for $j = 1, \ldots, \lambda_k$ is taken. Thus, $R_k = S_k \cdot T_k + N_k$ defines the sub-model for the appearance of one individual ring (fig. 8.20).

The third sub-model deals with the interaction of the neighboring rings. First, the rings are successively added into the model, according to their order by using a recursive function $f(R_1, \ldots, R_k) = L_k \cdot R_k + (1 - L_k) \cdot f(R_1, \ldots, R_{k-1})$ for $k = 1, \ldots, n$ and $R_0 = 0$ (fig. 8.21). Since every part of the surface is processed several times, all associated rings must be taken into account with most of the weight on the temporally last rings. A convex combination is used therefore. Because the surface is more changed by deeper indentations, a linear function $L_k : \mathbb{R}^2 \to [0, 1]$ is taken instead of a global parameter, which gives more weight to the front part of the ring than to the back. It is zero for points outside the ring. Values at the front and back of the rings are given by $a_k \sim \mathcal{U}([a^{\min}, a^{\max}])$ and $b_k \sim \mathcal{U}([b^{\min}, b^{\max}])$.

A summary of the parameters needed for the model is given in table 8.2. Some parameters are known explicitly (section 8.3), while the others are estimated by visual comparison using high resolution topography measurements (fig. 8.13a). Different simulations of the model using distinct parameter configurations are shown in fig. 8.22. In the end, height values of the preliminary texture image $\mathcal{T}_{\text{pre}}$ are adapted so that their distribution resembles that of the corresponding measurement $\mathcal{M}$. Their sample mean values $\hat{\mu}_\bullet$ and sample variances $\hat{\sigma}_\bullet^2$ should coincide, which is obtained by $\mathcal{T} = \sqrt{\hat{\sigma}_\mathcal{M} / \hat{\sigma}_{\mathcal{T}_{\text{pre}}}} \cdot (\mathcal{T}_{\text{pre}} - \hat{\mu}_\mathcal{M}) + \hat{\mu}_{\mathcal{T}_{\text{pre}}}$.



(a) Sub-model for ring appearance using $\phi_k = 0$, from left to right: S_k, T_k, N_k, R_k.



(b) 1d intersection of $S_k \cdot T_k$.

Figure 8.20: Illustration of sub-model for ring appearance with explanation of its parameters.

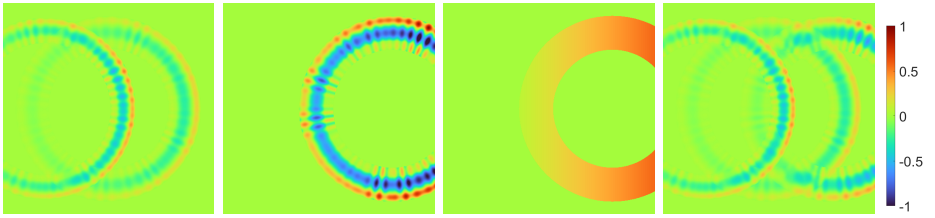
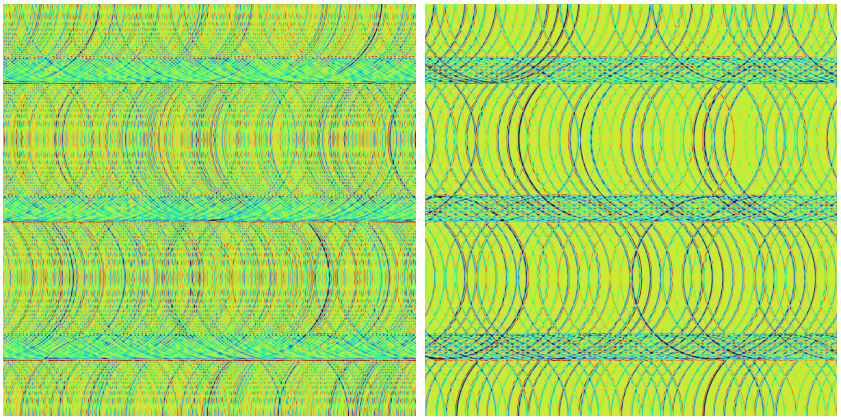
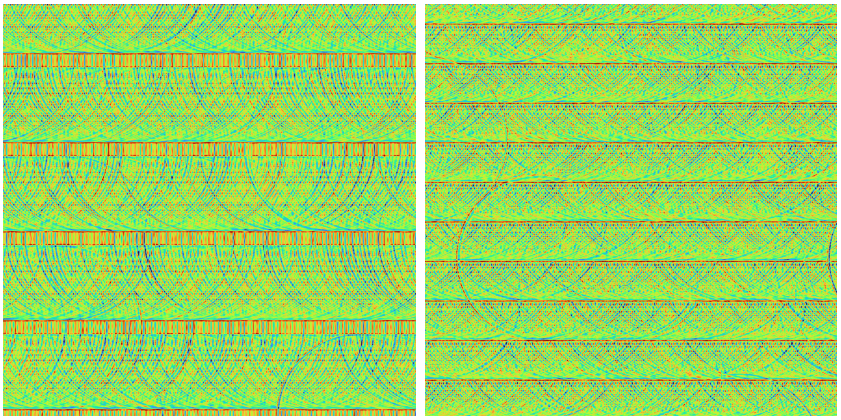


Figure 8.21: Sub-model for rings' interaction using $\phi_k = 0$, from left to right: $f(R_1, R_2)$, R_3, L_3, $f(R_1, R_2, R_3)$.



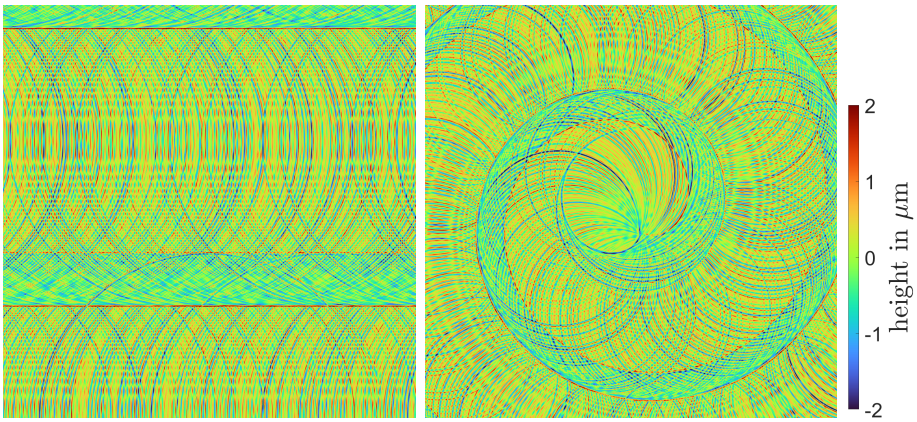
Parallel milling using $d = 4$mm, $\alpha = 0.2$, $\delta = 0.09$mm.

Parallel milling using $d = 4$mm, $\alpha = 0.2$, $\delta = 0.25$mm.



Parallel milling using $d = 4$mm, $\alpha = 0.5$, $\delta = 0.09$mm.

Parallel milling using $d = 4$mm, $\alpha = 0.8$, $\delta = 0.09$mm.



Parallel milling using $d = 8$mm, $\alpha = 0.2$, $\delta = 0.09$mm.

Spiral milling using $d = 4$mm, $\alpha = 0.2$, $\delta = 0.09$mm.

Figure 8.22: Adapted simulated milled surfaces generated with different parameter configurations. Imaged region is 10mm × 10mm.

	notation	definition	meaning	distribution	parameter	values
pattern	c_k	rings' center points	determined by tool path	$\mathcal{N}(\bar{c}_k, \Sigma_c)$	$\Sigma_c \in \mathbb{R}^{2\times2}_{>0}$	$(\Sigma_c)_{ii} = 1/300\delta,\ i = 1,2$
	d	diameter of ring	**diameter of milling head**		$d \in \mathbb{R}_{>0}$	$d \in \{4, 8\}$mm
	α	defines ρ (distance between neighboring tool paths)	**amount of overlap of neighboring tool paths**		$\alpha \in (0, 1)$	$\alpha \in \{0.2, 0.5, 0.8\}$
	γ	increase distance between neighboring tool paths	distance between blade and outer edge of tool		$\gamma \in (0, \alpha)$	$\gamma = 0.04$
	δ	distance between center points	depends on feed rate and tool's rotational speed		$\delta \in \mathbb{R}_{>0}$	$\delta = 0.09$mm
	ϵ	amount of rings with changed order			$\epsilon \in [0, 1]$	$\epsilon = 0.01$
appearance	w_k^-	width of indentation	width of cutting edge	$\mathcal{N}(\mu_{w^-}, \sigma_{w^-})$	$\mu_{w^-} \in (0, d/2)$ $\sigma_{w^-} \in \mathbb{R}_{>0}$	$\mu_{w^-} = 0.05$mm $\sigma_{w^-} = 1/6\mu_{w^-}$
	w_k^{+i}	width of inner accumulation	depends on edges' sharpness	$\mathcal{N}(\mu_{w^{+i}}, \sigma_{w^{+i}})$	$\mu_{w^{+i}} \in [0, d/2 - \mu_{w^-})$ $\sigma_{w^{+i}} \in \mathbb{R}_{>0}$	$\mu_{w^{+i}} = 0.025$mm $\sigma_{w^{+i}} = 1/30$
	w_k^{+o}	width of outer accumulation	depends on edges' sharpness	$\mathcal{N}(\mu_{w^{+o}}, \sigma_{w^{+o}})$	$\mu_{w^{+o}} \in \mathbb{R}_{\geqslant 0}$ $\sigma_{w^{+o}} \in \mathbb{R}_{>0}$	$\mu_{w^{+o}} = 0.025$mm $\sigma_{w^{+o}} = 1/30$
	ϕ_k	tilting direction	depends on tool path		$\phi_k \in (-\pi, \pi]$	computed by c_k
	$l_k^-,\ h_k^-$	minimal/maximal scaling of indentation depth	cutting depth with tilting	$\mathcal{N}(\mu_{\bullet^-}, \sigma_{\bullet^-})$	$\mu_{\bullet^-} \in \mathbb{R}_{\geqslant 0}$ $\sigma_{\bullet^-} \in \mathbb{R}_{>0}$	$\mu_{l^-} = 0.7,\ \mu_{h^-} = 1$ $\sigma_{\bullet^-} = 8/30$
	$l_k^{+i},\ h_k^{+i}$	minimal/maximal scaling of inner accumulation height	depends on edges' sharpness	$\mathcal{N}(\mu_{\bullet^{+i}}, \sigma_{\bullet^{+i}})$	$\mu_{\bullet^{+i}} \in \mathbb{R}_{\geqslant 0}$ $\sigma_{\bullet^{+i}} \in \mathbb{R}_{>0}$	$\mu_{l^{+i}} = 0,\ \mu_{h^{+i}} = 0.2$ $\sigma_{\bullet^{+i}} = 1/6$
	$l_k^{+o},\ h_k^{+o}$	minimal/maximal scaling of outer accumulation height	depends on edges' sharpness	$\mathcal{N}(\mu_{\bullet^{+o}}, \sigma_{\bullet^{+o}})$	$\mu_{\bullet^{+o}} \in \mathbb{R}_{\geqslant 0}$ $\sigma_{\bullet^{+o}} \in \mathbb{R}_{>0}$	$\mu_{l^{+o}} = 0.2,\ \mu_{h^{+o}} = 0.5$ $\sigma_{\bullet^{+o}} = 1/6$
	λ_k	number of sine curves		$\mathcal{P}(\lambda)$	$\lambda \in \mathbb{N}$	$\lambda = 50$
	τ_{k_j}	frequency of sine curves		$\mathcal{P}(\tau)$	$\tau \in \mathbb{N}$	$\tau = 50$
	ξ_{k_j}	shift of sine curves		$\mathcal{U}((-\pi, \pi])$		
interaction	a_k	minimal value convex combination		$\mathcal{U}([a^{\min}, a^{\max}])$	$a^{\min} \in [0, 1]$ $a^{\max} \in [a^{\min}, 1]$	$a^{\min} = 0$ $a^{\max} = 0.3$
	b_k	maximal value convex combination		$\mathcal{U}([b^{\min}, b^{\max}])$	$b^{\min} \in [0, 1]$ $b^{\max} \in [b^{\min}, 1]$	$b^{\min} = 0.1$ $b^{\max} = 0.4$

Table 8.2: Overview of all quantities needed for the milling model including their required parameters and choices thereof. Known quantities are highlighted (section 8.3).

8.6 Defect Modeling

8.6.1 Related Work

Defects are flaws on the surface (fig. 8.8, fig. 8.10) of an object that can range from geometrical defects (e.g., mechanical - dents, scratches, cracks, etc.) to material defects (e.g., oil, chemicals, etc.) as discussed by Black et al. [2]. The development of machine vision solutions for defect recognition requires image data of products containing various defects (Beyerer et al. [5]). Straightforward ways to generate defected image data are to either embed defects on defect-free real images or to train GAN models. First, a more classical, approach was presented by Haselmann et al. [286] which uses a multi-step stochastic process which generates defect textures that are added to defect-free surfaces. Although controllable, the method generates defects as 2D patches which cannot correctly represent light reflection disabling multi-view image generation and often causing inconsistent light appearance between the defected and correct (original) surface. More popular defect image synthesis using GAN is discussed by Niu et al. [221], Wang et al. [256] and Zhang et al. [255]. However, GAN-based methods cannot generate edge-case scenarios if they were not present in the sample dataset or multi-view representation of the same defect with varying acquisition conditions. Additionally, GANs are known to hallucinate [287], which can cause the results to introduce uncontrolled error representation into the generated dataset.

Since our focus is rule-based image synthesis, more suitable, algorithmic approaches are presented by Barisin et al. [231] and Jung et al. [336] for the simulation of concrete cracked surfaces. Although they give impressive results, their work is focused on 3D voxel data, while our focus is on metal surfaces with surface defects. Defected metal surface modeling was presented by Raymond et al. [207] where scratches were modeled by stacking layers with different scratch distributions. Unfortunately, this model does not represent curving scratches nor dents or scratches with varying geometrical depth. In this work, we use the parameterized methods presented in Chapter 6, capable of producing surfaces containing geometrical dent and scratch defects ensuring correct light scattering.

8.6.2 Defect Modeling Pipeline

For the sake of completeness, we first provide a short overview of the defecting workflow presented in Chapter 6. Three main steps of the defecting workflow are (1) defect positions generation, (2) generation of 3D geometries representing the defecting tools and (3) imprinting the defecting tools into the surface and creating geometrical masks which are used later for generating defect annotations. Defect positions are found by sampling points on inspected product geometry using various methods ranging from uni-

form to weighted ones. Two main defecting tools that are introduced are negatives of dents and scratches. Denting tools are typically spherical and parametrized for controlling their scale, rotation and shape. Scratch tools are created by first walking on a product geometry resulting in a path which is then converted into a solid geometry. Scratch tools are also parametrized enabling the control of their size and shape such as thickness and curviness. Imprinting defecting tools into inspected product mesh is performed using Boolean difference operation to ensure correct light response under varying view and light conditions. After defect imprinting, geometrical masks are created using inspected product mesh and defecting tool. The resulting geometry - a mask - can be a solid filling the defect or represent a shell covering the defect.

After the defecting workflow overview, we describe how it is used for defect modeling in our work. We focus on dent and scratch defects introduced in Chapter 6 to replicate physical defects described in section 8.3. First, positions are generated on the inspected product shape where the defects will take place. Since generating positions is parameterized, we can control where the defects will appear and the number of defects that will be created. To control where the defects will appear, constrained sampling is used so that only faces A, B, and C (fig. 8.11a) are sampled for positions. Once defecting positions are generated, defecting tools are generated and placed on found positions. As the generation of defecting tools is completely parameterized we can control their shape, size and orientation as discussed in Chapter 6. This enables the controllable creation of defecting tools that represent dent and scratch negatives that can be found on physical objects. Once generated and placed, defecting tools are imprinted into the inspected product surface altering the actual geometry. In this step, a geometrical object, a mask, fitting the created defect is further created using the defecting tool. The created mask object is used during the image synthesis step for the automatic generation of defect annotations, alongside realistic renders.

Described defecting process, both the mesh sampling algorithm for defect positioning as well as defecting tool creation are parameterized. This means that this approach enables the creation of defects whose parameters correspond to the shape, size and position of real defects. Therefore, we can create realistic defects resembling the real defects. That said, measured defect topographies are used to determine the defecting parameters. However, the defecting approach used in this work can be further extended so that defecting tools are created based on the defect topography information. Finally, this approach enables the creation of an arbitrary number of defected 3D objects. That leads to creating an arbitrary number of synthetic images of defected objects (fig. 8.5, fig. 8.6) with automated and pixel-precise annotations (fig. 8.7, fig. 8.9, fig. 8.10).

8.7 Dual Dataset

Inspection datasets of defects are difficult to obtain due to physical costs of producing defected parts, low probability of defects appearing and proprietary nature of the data. However, several datasets exist in the scientific community and provide an invaluable source of data for scientific advancements in the field. Publicly available datasets for inspection of metal objects [228] mostly focus on the defect recognition over flat surfaces and single-view inspection setups. While this simplifies the light response diversity of defect and surface geometry to ease defect recognition, it poses the danger of lower recognition of some defects due to low defect visibility under that specific setup. The aforementioned datasets contain objects such as hot rolled steel [245–248], slightly curved metal pads [236,337], rails [251] and pipes [250]. Datasets of smaller metal objects with complex geometry are also present, however also under single-view inspection setups [243,253]. More complex inspection setups are present in recent datasets by including single-axis rotation [338], multiple illumination directions [237] and the multiple view directions [339]. Multi-view inspection setups inspect the same surface multiple times while varying at least the direction of the incoming light source or the direction of the camera to reduce the chance of missing defects due to low visibility. Synthetic datasets for inspection exist in form of generic stochastic textures [340], outdoor maintenance scenes [224], multi-object pose estimation task [260], industrial part recognition [341] and surface defect recognition [339].

We present a dual dataset consisting of a real and synthetic part, following the multi-view inspection setup described by [339]. Our dataset focuses on recognition difficulties arising from complex reflectance of different surface finish patterns which may cause defect curtaining.

8.7.1 Real Dataset

The real dataset was acquired using a setup consisting of a robot manipulator, matrix grayscale camera, diffuse ring light and an acquisition table. The setup uses a thick black curtain to remove any influence of the environment illumination or reflection from within the acquisition box. The manipulator is used to position the camera and illumination onto predefined viewpoints. The acquisition table is a flat surface covered in diffuse black velvet and the acquired object is manually placed on top. The acquisition viewpoints are defined using V-POI [188] as an inspection plan, relative to the 3D model of the acquired object. Our acquisition plan consists of viewpoints focused on the middle of each inspected object face, at angles 0, 10 and 20 degrees from the respective surface normal visualized in fig. 8.23.

In order to minimize the placement deviation between the expected and actual position of the object, the operator must perform the *object position-*

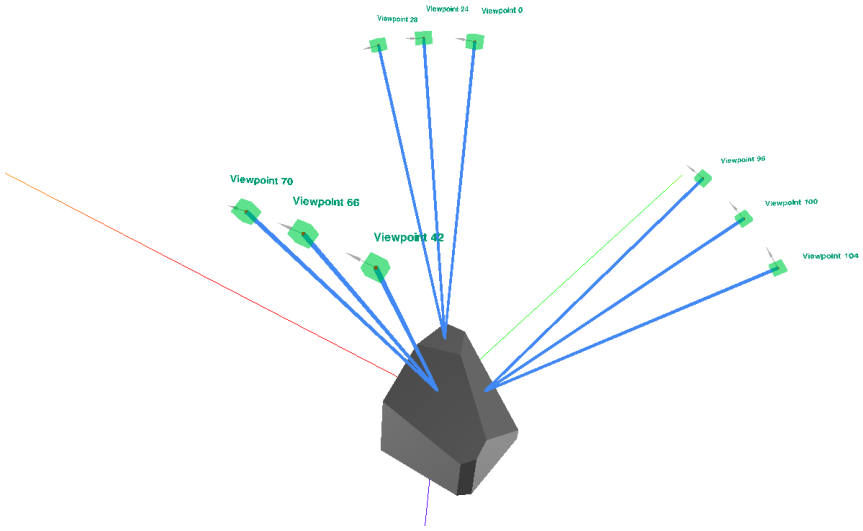


Figure 8.23: Visualization of viewpoints used for inspection.

ing routine. The positioning routine uses a predefined reference viewpoint and synthetic image of the object taken from it. In this case, it is the viewpoint placed perpendicularly to the C-face of the object. The physical camera is placed into position defined by the reference viewpoint and the reference synthetic image is overlayed. For research purposes we ensure that the real images are closely aligned with their synthetic correspondents by manually aligning the object to make the observed image match a synthetic reference image. For this procedure to be appropriate - either the synthetic image must contain image distortion introduced by the lens or the real image must be undistorted.

The real dataset consists of images of test bodies defined in section 8.3, acquired using the setup above and manually annotated using LabelMe [265]. In total, the dataset contains 30 objects, with 20 of them defected, and 12 viewpoints per object. Overall this amounts to 360 annotated images, 240 of which display defects. The dataset was split into train-val-test sets. The test set contains the 10 correct objects and 10 defected objects. Train and validation sets contain the remaining 10 defected objects, where only 1 viewpoint per object is used for the validation set.

8.7.2 Synthetic Dataset

The synthetic dataset generation environment described in section 8.2 was developed to match the physical acquisition setup. Dataset generator is an automated procedure which uses object, texture and defect specifications for generating annotated synthetic data in multiple stages. The first stage of dataset generator is geometric defecting as described in section 8.6, which

Parameter	Small dent	Big dent	Flat scratch	Curvy scratch
Quantity	5	3	2	2
Diameter	[0.02, 0.2]	[0.2, 1.0]	[0.02, 0.2]	[0.02, 0.1]
Elongation	[1, 2]	[1, 4]	-	-
Depth	[0.05, 0.2]	[0.2, 1.0]	-	-
Path length	-	-	[5, 20]	[10, 20]
Step size	-	-	0.1	1.0

Table 8.3: Defect specifications and ranges for uniform sampling.

generates and applies defects to the original object geometry. The defecting consists of sampling points on the object's surface, generating random defect geometries based on user specification and applying them on sampled locations. Depending on their specifications, defects are grouped into classes which are later used to obtain labels for recognition model development. Along with application of defects to the object surface, the defect geometry is grouped and stored in two ways: per-class for generating semantic segmentation masks and as separate geometries for generating instance segmentation masks. Defecting is performed in an iterative manner for each new defect specification, for a user-specified number of times, each time resulting in new instance of defected geometry. We define two types of defects: dents and scratches. To obtain an approximate defect specification ranges we measure the defects present in real objects. We further gently increase the range to generate unobserved defect shapes. The final defect generation specifications are provided in table 8.3. The defect specifications of dents and scratches are split into 2 sub-types to ensure that both types are always present in the images, however they are later treated same by the recognition model.

The second stage is done in parallel and produces texture images of the object surface using methods introduced in section 8.5. Like defects, the texture parameters are specified by ranges which are randomly sampled for every new image, thus introducing controlled surface appearance variation within desired ranges. For dataset generation we use parameters introduced in table 8.2 with increase in variation by sampling parameters presented in table 8.3. We empirically decided on this set of parameters as they model stochastic elements of the milling process and produced observable changes in textures. The ranges were chosen to model plausible effects on textures, such as the milling at different translational speeds or amplitudes of parallel and orthogonal tool vibrations.

The third stage combines the geometries and textures to produce annotated images, as described in section 8.2. Synthetic images and masks are generated for every instance of the defected geometry based on a predefined inspection plan (camera/light acquisition setup). An image-mask pair

Parameter	Set of values to randomly sample from
Ring flip probability (ϵ)	0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5
Tool path noise (Σ_c)	0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8
Ring separation (δ)	1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2
Ring width noise (σ_{ω^-})	0.3, 0.4, 0.5, 0.6, 0.7
Noise depth ($\sigma_{l^-}, \sigma_{h^-}$)	0.4, 0.6, 0.8, 1.0, 1.2
Sin curves number (λ)	30, 50, 70

Table 8.4: Texture variation parameters, modifying the default values in table 8.2.

is produced for all viewpoints and all geometries. To simulate the variety in production of surface finishes, for each surface of the object geometry a texture instance is selected at random, the texture is rotated randomly between $[-15, 15]$ degrees and translated by 5, 3 or 1 for faces A, B and C respectively. High sensitivity of textures to direction of light and camera ensures that the random texture shifts will produce greatly different appearances. Since the defective geometries have multiple instances, the correct object instances are generated by using original object geometry together with randomly sampled texture instances, offsets and rotations. To model different oxidation levels, we use the material roughness parameter which we randomly sample for each synthetic object from range $[0.05, 0.3]$.

The synthetic dataset is an approximate model of the production variance observed in the real dataset. Similarly to the real dataset, it contains 10 objects with matching textures. However, it contains 30 instances of correct and 30 instances of defective geometry per-object. The correct geometry instance is shared within each correct object instance and only the textures are applied uniquely to produce variety. The defected geometry instances are shared among objects to remove result variance due to geometry in the later experiments. In total, the dataset contains 31 unique geometries forming 600 unique object instances. Each object instance is inspected using same 9 viewpoints used in the real dataset totaling in 5400 images, 2700 of which contain defects. The images were rendered using 256 spp and 8 light bounces at resolution 1224×1025. The train-val-test set was produced in the ratio 70:10:20, by splitting along the dimension of object instances to ensure the same label distribution across all texture groups.

8.8 Quality Estimation

Dataset quality for machine learning is a field focusing on estimation of factors that affect recognition model development. Various metrics were proposed in recent works [342] to estimate the factors that affect the dataset quality, such as annotation quality or class balance. However, these met-

rics usually ignore the domain-specific and cross-domain elements that can produce various domain-specific biases, such as labeling sufficiency, domain task compatibility or domain coverage.

Quality estimation of a synthetic dataset should be measured by how accurately it models the real process of data acquisition, as well as how much utility does the recognition model get from it. It should give insight into the domain gap present between the two datasets to assist with further improvements of the synthetic dataset. To organize the different ways of measuring the quality of a dataset, we propose a distinction between *a priori* and *a posteriori* similarities. To measure the qualities of our synthetic data, we assume the existence of a representative dataset collected in the real world under controlled conditions. To the best of the authors knowledge, this is a first evaluation of synthetic data quality which goes beyond the comparison of recognition model performances.

8.8.1 A Priori Similarities

A priori quality estimation collects requirements which should be satisfied by the synthetic dataset to model the real acquisition process prior to data generation itself. For visual surface inspection, these requirements are tightly coupled with the results of physical processes used to manufacture and inspect the object. Measuring such a quality is difficult as we work with observations which aren't necessarily quantifiable by themselves. However, listing the assumptions taken when modeling the dataset provide valuable information on which effects might not have been modeled and could later affect the recognition model. Thus, the *a priori* quality of the dataset is proportional to the number and precision of the environment model used to generate the images.

8.8.2 A Posteriori Similarities

A posteriori quality estimation is done by comparing the similarity of the synthetic dataset to the collected real dataset. In this work we accumulate the results of metrics that compare the similarity between the corresponding dataset images (domain similarity) and the similarity of the dataset distributions through the training of defect recognition models on the required task (task similarity). The *a posteriori* quality of the dataset is proportional to the distances between datasets reported by the dataset similarity metrics.

Domain similarity

describes the dataset quality through the similarity of intensities and patterns in corresponding images between datasets. This type of dataset similarity is related to the amount of covariate shift. Covariate shift is a type

of domain shift where the conditional (posterior) distributions are equivalent between distributions $p(y|x)$, but the prior (image) distributions differ $p(x)$ [343]. Since the image space is high-dimensional and sensitive to offsets in the scene, a more efficient comparison can be performed through an image descriptor which summarizes some aspects of an image into a short feature vector used for comparison.

Value-wise similarity compares the global distribution of intensities between the images with corresponding viewpoints (e.g. histogram distances). This type of comparison is affected by scene media that are spatially invariant, e.g. illumination strength and light absorption of a material.

Pattern-wise similarity compares the spatial patterns between the images. This type of comparison is affected by the scene structures, e.g. patterns of the surface finish, light source shape and material reflectance function.

Handmade image descriptors focus on special aspects of patterns in images, such as: pixel-wise closeness and alignment (e.g. mean absolute error, PSNR) or statistical similarity (e.g. SSIM [344]). Leveraging learned feature extractors for feature-space comparison (e.g. Inception score [345], FID [346], LPIPS [347]) alleviate the need of designing manual image descriptors by instead relying on a large and diverse dataset and training of a deep feature extractor to model a general-purpose feature space that can work across different domains of images.

Multiple distance measures are available in the field of domain adaptation for measuring different shifts between data distributions [343], where the goal is to reduce the distance between two or more domains of a learned feature extractor. These methods are also used in generative adversarial models, where a feature matching loss function is used to improve the realism of the generator model by minimizing the distance in a feature space modeled by the discriminator model [348].

Task Similarity

describes the distance between the data distributions through the conditional dependence of label space on image space. This type of dataset similarity is related to the amount of concept shift present between the datasets. Concept shift is a type of domain shift where the conditional (posterior) distributions $p(y|x)$ differ between domains, while the priors $p(x)$ are the same [343]. In the field of domain adaptation [343], a common approach to measure the effectiveness of such methods is through the difference in results of task metrics when a model is trained on the source dataset and when such methods are used to adapt it to the target domains. As discussed in [349], this approach measures the difference in biases present in the two datasets, which can be interpreted as domain shifts when working with datasets acquired from different environments.

8.9 Pipeline Evaluation

8.9.1 A Priori Similarities

Using physics based light simulation we ensure that the images are produced in a realistic way (section 8.2.7). Using the CAD model of the object we guarantee the produced images restrict the defect recognition task to the target object thus allowing development of simpler recognition models (section 8.2.5). Using textures modeled to match the real surface measurements we further restrict the recognition task to ignore nominal patterns (section 8.5). Texture parameters around the observed nominal values increase model robustness towards ignoring the surface texture and focus more on defect texture (section 8.7.2). Additional defect randomization parameters that contain ranges of expected defect dimensions and shapes, we restrict the model to safely recognize observed defects, but also defects outside the observed range for better generalization outside the real dataset set (section 8.7.2).

8.9.2 A Posteriori Similarities

Domain Similarity

To compare the synthetic images to real ones we need to remove the influence of background noise and minor offsets, present from the imprecision of the acquisition system and background scene structures. Thus we first mask out all the regions that do not contain the observed object face containing the target texture. The polygonal target face was manually annotated and a common meta-mask was constructed from the intersection of all masks for that viewpoint, thus removing the influence of offsets in hard edges between the face polygon and the black background. Real image histograms have a region of zeros after intensity 0 which we simulate by adding the mean gap size across all real images for a particular object. Next, we estimate a multiplication factor that compensates for the illumination intensity and material absorption differences in synthetic images. The factor can be extracted from the integral of the rendering equation [75] under the assumption that the environment reflectance and the inter-reflectance of the surface micro-structures is negligible, which holds since we use a controlled environment and normal-mapping respectively. The estimated factors and biases are: (0.632, 34) for sandblasted, (1.481, 39) for parallel and (1.526, 33) for spiral milling. Finally, as the synthetic pinhole camera model does not include blurring and imperfections of the real camera, we artificially blur the synthetic images using defocus blur [350].

When evaluating the metrics, we select the smallest (closest) value between synthetic object instances for the corresponding viewpoints. We average the similarities of images across multiple viewpoints for objects sharing

Method (↑)	Sandblasted	Parallel	Spiral	All
Histogram (Pearson)	0.840	0.794	0.772	0.794
1 - MAE	0.807	0.681	0.655	0.696
SSIM	0.896	0.561	0.585	0.638
1 - LPIPS	0.916	0.660	0.686	0.722

Table 8.5: Domain similarities averaged within and across texture types. MAE and LPIPS were inverted to measure the degree of similarity.

the same texture type. The results across different similarity metrics are reported in table 8.5. To summarize the distance between normalized intensity histograms we use the Pearson correlation coefficient, which is bounded between $[-1, 1]$. Mean absolute error (MAE) and Structural similarity (SSIM) are bounded between $[0, 1]$ and were used to measure the average difference in values and similarity of statistics of values, respectively. For the learned descriptor LPIPS we used a stock pre-trained AlexNet [351] backbone, which enforces no restrictions on image size and returns the similarity bounded between $[0, 1]$.

Task Similarity

To evaluate task similarity, we train a UNet [268] with a ResNet-50 [270] backbone. For each experiment we train 5 models with random parameter initializations and report the top obtained result. Since the number of non-backgorund pixels is highly unrepresented, we observed the need to use class weighting which we empirically found to be $(1, 1.5, 1.5)$ for the real and $(1, 2, 2)$ for the synthetic dataset. The model is trained to perform semantic segmentation through pixel classification into classes: background, dent, scratch. We optimize models using the AdamW optimizer [271] with learning rate 10^{-4} and L_2 weight regularization of 10^{-5}, for a maximum of 1000 epochs with early stopping at validation loss convergence upon 5 consecutive validations without at least 10^{-2} relative improvement. For model fine-tuning we decrease the learning rate to 10^{-5}. For memory efficiency, during training we extract image crops of size 256×256 and during evaluation we split the images into a 3×3 grid of patches of size 416×352. All images were zero padded to ensure dimension divisibility correctness when evaluating the U-Net and image patching. To speed up model training, we center the images linearly between $[-1, 1]$.

We use the images pre-processed as described in section 8.9.2. Since rendering images is a memory and compute expensive operation we offload as much of the domain alignment as possible to online post-processing. To simulate the non-zero background, we artificially add Gaussian noise of amplitudes between $[0, 5]$ to the background around the object. To simulate the vertical blooming effect caused by photon leakage within the saturated

regions of the CCD sensor array, we extract the maximum value of each image column, mask it using a threshold 0.95, blur it using a box filter of size 64, multiply by 0.02 and add to the original image with clipping. Manually annotated real images contain imprecision in masks as they often cover the object surface around the rim of the defect. We emulate this by dilating the masks by 1 pixel. The synthetic images contain masks in regions that are mostly uniform and should be considered invisible. We pre-filter the masks to remove defect with insufficient visibility. We empirically estimate the visibility as the difference between the mean intensity of the defect and the mean intensity of the surface ring around the defect, calculated as difference between the 2 times dilated and the original defect mask. Defects are considered visible if the difference in means is over 0.05.

Data augmentation further increases the diversity of the dataset and helps regularizing the recognition model to consider the important features. To simulate the offsets of the camera we use a random rotation of amplitude 15 degree. Illumination strength and object material diffusion variation can be jointly controlled using random exposure from interval $[-0.5, 1.5]$. To simulate the blurred background structures and impurities in the lens or the light carrying medium, we add Gaussian noise of amplitude between $[0, 5]$ to the entire image and then blur it using defocus blur [350] of kernel sizes between $[1, 3]$. Gaussian noise of amplitude between $[0, 5]$ is added once more to simulate the static noise of the real images. Finally, images can be randomly flipped horizontally to increase the texture-structure pair diversity. All of the augmentations are performed with probability of 50%. Finally, we perform intensity biased random cropping [339] using threshold 15 to ensure production of crops that contain more visible structures in the image.

The results are collected in table 8.6. When computing the mean score, the contribution of the background class was ignored as it is the majority class that would skew the results away from the defect classes. To evaluate the concept shift, we compare common semantic segmentation metrics of the model generalization when trained on the real subset to the synthetic subset.

8.10 Discussion

8.10.1 Pipeline Controlability and Simplicity

The pipeline is very versatile and allows generation of a wide variety of texture variations needed for domain randomization. Parametrization of the modules allows defining elements using physical measurements. The process is relatively fast and its modularity allows faster iteration and extensibility to new textures and defects. The biggest bottleneck is the rendering process which slows down the dataset generation.

Texture	Domains	mP	mR	mF1	mIoU
Sandblasted	Sy → Sy	57.0	53.1	54.7	37.7
	Sy → Re	34.0	64.3	41.7	26.3
	Sy+FT → Re	60.1	71.9	65.5	49.6
	Re → Re	57.9	61.9	59.1	42.6
Parallel	Sy → Sy	57.1	39.6	45.6	29.6
	Sy → Re	26.5	23.0	23.5	13.8
	Sy+FT → Re	52.6	33.4	38.3	23.9
	Re → Re	50.3	34.5	40.8	25.8
Spiral	Sy → Sy	59.5	39.7	47.6	31.3
	Sy → Re	35.2	22.1	26.4	15.5
	Sy+FT → Re	49.5	40.1	43.9	28.4
	Re → Re	50.6	43.1	46.2	30.7
All	Sy → Sy	59.4	46.0	51.5	34.7
	Sy → Re	32.2	31.3	31.6	18.9
	Sy+FT → Re	59.0	49.1	53.1	36.1
	Re → Re	59.0	48.2	52.6	35.8

Table 8.6: Task similarities between texture groups. The domains column symbolizes the experiment *training* → *testing* domain, with the real (Re) and synthetic (Sy) domains. Fine-tuning (FT) is performed using real data after training on synthetic data.

Currently, the large number of parameters makes it difficult to study the influence of different parameter sampling schemes on the recognition performance. However, this pipeline opens the opportunity to design a higher level of abstraction for dataset design through modelling of lower dimensional constraints over larger sets of parameters. This study is however out of scope and left a future work.

8.10.2 Domain Similarity

Measuring texture similarity is a difficult problem due to surface and acquisition imperfections present in the real data. Camera blurring, surface imperfections and other effects move away from the nominal texture without degrading its performance, thus producing falsely bad results. In addition, it is difficult to generate a nominal image that perfectly aligns to the acquired image due camera-object and texture-object alignments. In this work we instead rely on methods that compare global or local pixel statistics of textures to remove alignment. From results in table 8.5 we observe that the sandblasted textures have the highest degree of similarity compared to the parallel and spiral milling textures. This result is related to the simplicity of modeling the sandblasted texture as the texture appears as mostly uniform noise. More structured textures such as those of milling are more difficult to

replicate as they have significantly more parameters that interplay to produce unique patterns. The Pearson correlation of image histograms show relatively higher similarity values in milled textures, which is likely due to the histogram-based image adjustments performed in section 8.9.2.

Comparing the values of metrics is difficult, even though they are bounded between $[0, 1]$ and might measure similar properties of images, they do not share the same linearity of response to changes in inputs. A separate study is required to compare which physical properties different metrics attempt measuring over additional datasets. Since such a study targets a different task, it requires different definitions of dataset parameters which is out of scope of this study.

8.10.3 Task Similarity and Recognition Performance

The defect recognition results in table 8.6 show very similar performance in all experiments, except when the model trained on synthetic domain is evaluated directly on the real domain. The similarity of results, even after fine-tuning, indicates that the models are saturated in their capacity to solve the task. The low generalization to real domain indicates that there exists a misalignment between the synthetic and real data distributions, both in the form of covariate shift (as observed by similarity metrics) and concept shift. Results of models trained on all textures do not surpass the performance on the highest performing, sandblasted texture, leading us to the conclusion that training a model on multiple textures at once does not give any benefit over training a separate model for each texture.

When inspecting the model predictions over the datasets we notice some patterns. All models tend to predict masks that are slightly wider than the label. Most false positives come from the milled textures, in the lines where two milling path circles overlap and along the exit lines. In sandblasted textures, tiny glints from the surface are often falsely predicted as defected. False positives are also present in minor deviations in object shape such as edge scuffing, beveling, deeper sandblasting holes, etched signature, on paper label sticker and where the milled pattern has high contrast. Models also tend to predict defects on out-of-focus faces that the annotator left as unannotated due to heavy blurring or unrecognazibility. When predicting scratches, the models prefer areas where the texture has lower contrast and defect stand out. Some scratches that are not close to being horizontal or vertical are not predicted, even tho the visibility is similar to the predicted ones, meaning the recognition models need more filters in lower levels or more explicit rotation invariance. Overall, it seems a higher resolution might help the model distinguish between defect and texture patterns.

8.10.4 Influence of Domain Similarity on Task Performance

The largest drop in recognition performance is on milled textures, which is consistent with their lower domain similarity compared to sandblasted textures. Furthermore, SSIM and LPIPS seem to be linearly correlated to the best obtained recognition performance. If this were true, we could use these metrics when iteratively improving the synthetic data to reduce the domain gap at its source, the data simulation or generation process. This forms the questions of weather these metrics are indeed predictive of the domain gap that impedes the recognition performance, which other metrics have a similar property and why MAE and histogram comparison does not seem to be. These questions are vastly out of scope of this paper and we leave their treatment for a future study.

8.11 Conclusion

Specific requirements of machine vision for surface inspection require customized dataset and thus highly controllable context for realistic image generation. We presented a complete image synthesis pipeline which consists of surface topography measurements, parameterization and development texture models representing the measurements, simulation of image acquisition context for image synthesis and dataset generation. In collaboration with surface metrology experts test bodies were created containing industry-relevant, sandblasted and milling patterns. Surface topography was measured, analyzed and parameterized. Based on this requirements for bridging computer graphics surface modeling and surface metrology parameters were introduced. Resulting parameterized surface texture models ensured rich variation and controllability of surface texture patterns. The pipeline was put into practice and used to generate a dual synthetic and real dataset which is made publically accessible for further research. Quality estimation of the whole pipeline and synthetic data was performed. From experimental results, it can be concluded that further research is needed to quantify the domain gap and which elements of it affect the recognition models. Finally, the results indicated misalignment in synthetic and real data. This pointed to further research directions regarding the adaptation of the acquisition context, such as camera parameters, for image synthesis.

Core references

Juraj Fulir, Natascha Jeziorski, Tobias Herffurth, **Lovro Bosnar**, Thomas Gischkat, Katja Schladitz, Henrike Stephani, Claudia Redenbach, Hans Hagen and Petra Gospodnetić, "SYNOSIS: Image synthesis pipeline for machine vision in metal surface inspection", 2024. To be published.

Part IV

Conclusion

Chapter 9

Conclusion

The work is this thesis, as discussed in Section 1.1, is motivated by the virtual inspection planning and virtual environment introduced by Gospodnetic [3]. The inspection planning process includes the tasks of the placement of acquisition hardware for obtaining the required coverage as well as the development of defect recognition algorithms for acquired images. Planning represents a complex problem which must consider product shape, reflectance, texture and defects as well as light and camera. That said, the work in this thesis extends the virtual environment, using computer graphics methods, with an image synthesis pipeline as well as parameterized surface texture and defects. With this work, now it is possible to generate an arbitrary amount of photorealistic images which simulate images acquired by a system inspecting metal surfaces. This can be used for the evaluation of acquisition quality, i.e., estimating product surface coverage with respect to the illumination response of the surface, which wasn't possible until now. Further, this can be used for creating arbitrarily large datasets with realistic and varying textures as well as realistic and diversified defects coupled with pixel-precise annotations.

The aforementioned extensions to the virtual inspection planning environment are presented in the form of **thesis contributions**:

1. A novel image synthesis pipeline for generating synthetic data for surface inspection, introduced in Chapter 4 and further extended in Chapter 8

2. A procedural approach to parameterized surface texture modeling, introduced in Chapter 5

3. A procedural approach to parameterized defect modeling, introduced in Chapter 6

These contributions make it possible to obtain the required amount of representative product surfaces with realistic and varying surface textures

and realistic as well as diversified defects in a controllable and automated manner as discussed in Chapters 7 and 8. Now, the virtual environment [3] extended with these contributions makes it possible for inspection planning experts to perform the planning process virtually without computer graphics knowledge.

The first contribution are requirements for image synthesis for surface inspection planning and an image synthesis pipeline satisfying those requirements. With the presented pipeline, it is possible to synthesize realistic images of inspected product surfaces, both ideal and defected, in a controllable and automated manner. As such, the pipeline can be used for assessing the placement of acquisition hardware or for obtaining the required amount of defected image data for the development of defect recognition algorithms. This is particularly important for surface inspection tasks where data is difficult to collect or the labelling is too expensive to collect a large amount of it. Surface texture and surface defects are identified as crucial elements for inspection purposes and thus, they were the focus of the next two contributions.

The second contribution are requirements for texture synthesis for visual surface inspection and novel, parameterized texture synthesis models satisfying those requirements. The presented models are capable of generating circular, parallel and radial brushing texture patterns common in machined surfaces. As such, once applied on the product surface, the presented textures provide a physically relevant light response. This promises numerous applications in surface inspection planning. First, it enables assessing the surface coverage for particular configurations of acquisition hardware, completely virtually. Second, it enables assessing surface and defect visibility for particular view and light conditions.

The third contribution are requirements for defect modeling in surface inspection, parameterized workflow for positioning, generating and imprinting diversified defects on the product surface and automated generation of precise defect annotations. Presented models are based on procedural geometry, thus enabling parametric simulation of geometrical defects such as dents and scratches, which are two of the most frequent defect types. Defects are imprinted into product geometry ensuring realistic light behaviour. Finally, the presented methods generate additional defect information needed for pixel-precise defect annotation during the synthesis of defected product images.

9.1 Implications

This thesis provides a complete image synthesis pipeline based on computer graphics for virtual inspection planning. The requirements and components of image synthesis crucial for surface inspection are discussed. Further, this

work connects computer graphics procedural modeling of surface textures and defects with real-world surface topographies and manufacturing. As such it gives the foundation for further research regarding image synthesis and parameterized surface modeling for surface inspection and integration of manufacturing knowledge.

9.1.1 Image synthesis for Surface Inspection Planning

By having a virtual environment for inspection planning at their disposal, the crucial questions that inspection planning experts would like to answer are *what is the quality of particular acquisition hardware setup in terms of surface/defect visibility and how to configure it with minimal cost and effort?*. To enable answering those questions this thesis extends the virtual inspection environment [189] with an image synthesis pipeline for surface inspection. The introduced pipeline enables the synthesis of realistic images containing ideal and defected metal surfaces for a predefined inspection environment setup. The image synthesis is based on a controllable image acquisition context (i.e., a 3D scene containing inspected product, lights and cameras) and uses physically-based, global illumination simulation. This way, complete control over the simulated product surface, light and camera is available. This enables inspection planning experts to set up the desired parameters of the acquisition camera and light for evaluation and configuration. Further, configuring the desired defected product surface with geometrically imprinted defects is possible. Once the image acquisition context is defined, physically-based light simulation is used to ensure realistic surface and defect visibility under varying light and view conditions. That said, the presented image synthesis pipeline ensures the realistic representation of a virtual inspection environment as well as parameterized image synthesis. Therefore, inspection planning experts can completely virtually, visualize visible surfaces and defects for a given inspection plan, with minimal computer graphics knowledge. The presented image synthesis pipeline promises numerous applications besides acquisition hardware evaluation such as generation of well-balanced synthetic datasets for the development of defect recognition algorithms. These applications further motivated our research towards modeling product surface texture and defects.

9.1.2 Procedural Texture Synthesis for Surface Inspection

With the developed image synthesis pipeline for surface inspection, the focus was on further improving synthetic product images to be representative, i.e., as similar as possible to the images of real products. Since surface texture highly influences surface appearance, it is identified as a crucial element of inspected products for modeling. In this work, parameterized texturing models capable of reproducing circular, parallel and radial brushing texture

patterns, resembling common surface patterns resulting from machining, are introduced. Now, inspection planning experts can configure and thus simulate a wide range of varying surface textures to assess product coverage for a particular acquisition plan. This is possible since parameters are aimed to represent real-world surface parameters and not computer graphics surface parameters. The key observation that influenced texture synthesis research presented in this thesis is that the surface texture of inspected products is largely determined by the machining process. Therefore, the introduced models were developed by taking into account surface metrology concepts and connecting those with computer graphics procedural texture modeling. That said, it is important to highlight that synthesizing representative product surface textures relies on obtaining the real machining parameters and using them for modeling.

9.1.3 Procedural Surface Defects Modeling

Defect shapes and their occurrence on product surfaces are unpredictable. Therefore, a virtual inspection environment must be able to simulate products with diversified defect shapes, sizes and positions for the development of robust defect recognition algorithms. For this purpose, a parameterized workflow based on procedural geometry modeling consisting of three main steps: defect placement, defect negative modeling and defect imprinting on the product surface was introduced. Each step is parameterized enabling controllable placement and creation of diversified defect shapes. Dents and scratches are identified as common surface defects and thus particular focus was put on modeling their geometrical negatives. Since defects are modeled geometrically, the light response can be simulated correctly, which would not be the case if we only used normal mapping. Correct defect light response represents important information for determining defect visibility for different acquisition hardware setups. Further, using the modeled geometrical negatives, it was shown how precise defect annotations can be created automatically during rendering. This way, complete training data information can be created automatically, evading the cumbersome and subjective defect annotations. The resulting workflow is intended for intuitive use by inspection planning experts. This means that a wide diversity of defected products can be simulated with intuitive parameters and minimal computer graphics knowledge. With the introduced workflow, now it is possible to generate defects with diversified shapes and locations, over product surface, in a controllable and automated manner. These contributions are enabling the creation of arbitrary large image datasets containing defected products with diversified defects with pixel-precise annotations.

9.1.4 Synthetic Data for Defect Segmentation on Complex Metal Surfaces

Defect segmentation is crucial for defect recognition during automated visual surface inspection. Metal surfaces are particularly complex for inspection and the development of defect segmentation algorithms due to high reflectivity and anisotropic reflection caused by surface textures. Obtaining the required amount of defected product surface images is hard due to the low availability of defected product samples. Further, existing datasets are sparse, even more so when it comes to multi-view inspection and often focus on planar surfaces which are hard to generalize to more complex, curved surfaces. With the presented work, we showed that the introduced image synthesis pipeline, procedural texture modeling and procedural defect modeling can be used as a solution to those problems. We introduced a real and synthetic datasets pair for multi-view inspection and used it for training experiments. Obtaining the real dataset was limited by the number of defected physical products at our disposal. On the other hand, this was not the case with the synthetic dataset, where an arbitrary amount of defected products with varying textures and diversified defects could be generated. In this experiment, the presented methods for image synthesis, texture modeling and defects modeling are used in a realistic scenario and shown to be highly relevant due lack of training datasets and customized dataset creation. The experiment results showed that using custom synthetic image data, although hindered by the domain gap, is the most promising approach when the amount of real data is restricted.

9.1.5 Image Synthesis Pipeline for Machine Vision in Metal Surface Inspection

After the pipeline and parameterized surface texture and defects presented here were applied in the realistic scenario, we identified further research directions aiming to reduce the gap between real and synthetic images. The crucial task was developing texture models with even richer variations and more precise parameters to real surface parameters. The key difference in models introduced here before and developed now is that previous texture models are developed to *resemble the surface appearance as close as possible, without measurements*, while new texture models *actually follow machining parameters and measurements*. That required extending the pipeline with actual surface topography measurement and collaboration with manufacturing and metrology experts. For the purpose of this work, sandblasted and milled surfaces, which represent industry-relevant machining patterns, were created, analyzed, measured and parameterized. To bridge surface measurements and computer graphics modeling, requirements for developing stochastic geometry models for the representation of measurements are in-

troduced. Developed parameterized surface texture models ensure rich control over synthetic dataset generation and variation. The extended pipeline now encompasses physical surface measurement, parameterization and development of stochastic geometry models representing the measurements, applying developed textures on simulated products, image synthesis and dataset generation. The pipeline was used to create a dual synthetic and real dataset which and made publicly available. Performing quality estimation of the whole pipeline and synthetic dataset for machine learning defect recognition led us to the conclusion that further research is needed in terms of quantifying the domain gap and how much utility the recognition models get from it. Quality estimation of synthetic data was performed by training defect segmentation networks on synthetic and real data. The result of the experiment indicated a misalignment of synthetic and real data and pointed to further research in closing the domain gap. Therefore, besides surface texture pattern modeling, it is required to further adapt other elements of the acquisition context (e.g., simulated camera) for image synthesis.

9.2 Outlook

With this work, requirements, workflows, methods and implementations are introduced and available for realistic image synthesis for virtual surface inspection planning. The particular focus was put on procedural and thus parameterized models for surface textures and defects which are crucial for realistic image synthesis for inspection planning. Now is the time to focus research and development on further improvements of the presented approaches. First, as discussed, image synthesis highly depends both on 3D scene modeling and rendering. That said, further simulation of real camera parameters as well as different light shapes and intensity distributions must be researched and incorporated. Camera lens and sensor properties as well as light distribution modeling improvements will enable making virtual environment to resemble the real environment to an even higher degree. Second, since products are manufactured with a wide range of machining methods which are standardized and deterministic, the machining domain knowledge, as well as surface metrology, must be further incorporated into parameterized surface texture modeling. This way, it would be possible to design libraries of standardized, parameterized and physically correct surface textures. Equally important, the development of surface textures must be accompanied with further research on mapping the created textures on arbitrarily complex geometries with minimal manual work. Such developments would enable inspection planning experts to set up a virtual inspection environment for any manufactured product. Third, defects are unpredictable and highly diversified ranging from mechanical to chemical, surface, sub-surface, etc. Therefore, the categorization and standardization

of defects across different manufacturing processes is crucial. This way, the development of parameterized defecting models matching those standardizations would be possible. In order to do so, it is crucial to connect computer graphics research and manufacturing industry. This way, domain knowledge regarding actual and expected defects could be incorporated to develop generalized parameterized models. Once such a library becomes available, the development of robust defect recognition algorithms can completely rely on controllable and thus diversified synthetic image data generation. Such developments would enable the creation of arbitrarily large synthetic datasets which can be adapted to any inspection planning scenario.

Bibliography

[1] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-time rendering*. Crc Press, 2019.

[2] J. T. Black and R. A. Kohser, *DeGarmo's materials and processes in manufacturing*. John Wiley & Sons, 2020.

[3] P. Gospodnetic, "Visual surface inspection planning for industrial applications," Ph.D. dissertation, "TU Kaiserslautern", 2021. [Online]. Available: https://publica.fraunhofer.de/handle/publica/416618

[4] M. Mohammadikaji, "Simulation-based planning of machine vision inspection systems with an application to laser triangulation," Ph.D. dissertation, KIT, 2019.

[5] J. Beyerer, F. P. León, and C. Frese, *Machine vision: Automated visual inspection: Theory, practice and applications*. Springer, 2015.

[6] P. Gospodnetic, D. Mosbach, M. Rauhut, and H. Hagen, "Flexible surface inspection planning pipeline," in *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2020, pp. 644–652.

[7] P. Gospodnetić, M. Rauhut, and H. Hagen, "Surface inspection planning using 3d visualization," in *LEVIA'19: Leipzig Symposium on Visualization in Applications*, 2019.

[8] P. Gospodnetić, D. Mosbach, M. Rauhut, and H. Hagen, "Viewpoint placement for inspection planning," *Machine Vision and Applications*, vol. 33, no. 2, 2022.

[9] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces." *Rendering techniques*, vol. 2007, p. 18th, 2007.

[10] S. Deguy, "The new age of procedural texturing." 2019.

[11] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[12] Adobe, "Adobe Substance 3D Painter." [Online]. Available: https://www.adobe.com/products/substance3d-painter.html

[13] E. Lengyel, *Mathematics for 3D game programming and computer graphics*. Course Technology Press, 2011.

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

[15] B. T. Phong, "Illumination for computer generated pictures," in *Seminal graphics: pioneering efforts that shaped the field*, 1998, pp. 95–101.

[16] E. Catmull and J. Clark, "Recursively generated b-spline surfaces on arbitrary topological meshes," in *Seminal graphics: pioneering efforts that shaped the field*, 1998, pp. 183–188.

[17] H. H. Söderlund, A. Evans, and T. Akenine-Möller, "Ray tracing of signed distance function grids," *Journal of Computer Graphics Techniques Vol*, vol. 11, no. 3, 2022.

[18] S. Chen, R. Xu, J. Xu, S. Xin, C. Tu, C. Yang, and L. Lu, "Quickcsg-modeling: Quick csg operations based on fusing signed distance fields for vr modeling," *ACM Transactions on Multimedia Computing, Communications and Applications*, 2023.

[19] L. J. Tomczak, "Gpu ray marching of distance fields," *Technical University of Denmark*, vol. 8, 2012.

[20] T. S. Newman and H. Yi, "A survey of the marching cubes algorithm," *Computers & Graphics*, vol. 30, no. 5, pp. 854–879, 2006.

[21] K. Museth, N. Avramoussis, and D. Bailey, "Openvdb," in *ACM SIGGRAPH 2019 Courses*, 2019, pp. 1–56.

[22] S. Laine and T. Karras, "Efficient sparse voxel octrees–analysis, extensions, and implementation," *NVIDIA Corporation*, vol. 2, no. 6, 2010.

[23] C. Griwodz, S. Gasparini, L. Calvet, P. Gurdjos, F. Castan, B. Maujean, G. De Lillo, and Y. Lanthony, "Alicevision meshroom: An open-source 3d reconstruction pipeline," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 241–247.

[24] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–12, 2014.

[25] W. T. Reeves, "Particle systems—a technique for modeling a class of fuzzy objects," in *Seminal graphics: pioneering efforts that shaped the field*, 1998, pp. 203–220.

[26] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 335–342.

[27] S. Rusinkiewicz and M. Levoy, "Qsplat: A multiresolution point rendering system for large meshes," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 343–352.

[28] J. Dorsey, H. Rushmeier, and F. Sillion, *Digital modeling of material appearance*. Elsevier, 2010.

[29] P. Christensen, J. Fong, J. Shade, W. Wooten, B. Schubert, A. Kensler, S. Friedman, C. Kilpatrick, C. Ramshaw, M. Bannister *et al.*, "Renderman: An advanced path-tracing architecture for movie rendering," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 3, pp. 1–21, 2018.

[30] N. Hoffman, "Background: physics and math of shading," *Physically Based Shading in Theory and Practice*, vol. 24, no. 3, pp. 211–223, 2013.

[31] K. E. Torrance and E. M. Sparrow, "Theory for off-specular reflection from roughened surfaces," *Josa*, vol. 57, no. 9, pp. 1105–1114, 1967.

[32] J. Fong, M. Wrenninge, C. Kulla, and R. Habel, "Production volume rendering: Siggraph 2017 course," in *ACM SIGGRAPH 2017 Courses*, 2017, pp. 1–79.

[33] D. Guarnera, G. C. Guarnera, A. Ghosh, C. Denk, and M. Glencross, "Brdf representation and acquisition," in *Computer Graphics Forum*, vol. 35, no. 2. Wiley Online Library, 2016, pp. 625–650.

[34] J. Lambert, "Photometria sive de mensura de gratibus luminis, colorum et umbrae. eberhard klett, augsburg, 1760," *W. Engleman, Lambert's Photometrie. Leipzig*, vol. 127, no. 128, p. 2, 1982.

[35] W. McDermott, *The PBR Guide*. Allegorithmic, 2018.

[36] J. F. Blinn, "Models of light reflection for computer synthesized pictures," in *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, 1977, pp. 192–198.

[37] E. P. Lafortune, S.-C. Foo, K. E. Torrance, and D. P. Greenberg, "Non-linear approximation of reflectance functions," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 117–126.

[38] E. Heitz, "Understanding the masking-shadowing function in microfacet-based brdfs," *Journal of Computer Graphics Techniques*, vol. 3, no. 2, pp. 32–91, 2014.

[39] C. Schlick, "An inexpensive brdf model for physically-based rendering," in *Computer graphics forum*, vol. 13, no. 3. Wiley Online Library, 1994, pp. 233–246.

[40] N. Hoffman, "Fresnel Equations Considered Harmful," in *Workshop on Material Appearance Modeling*, R. Klein and H. Rushmeier, Eds. The Eurographics Association, 2019.

[41] M. Oren and S. K. Nayar, "Generalization of lambert's reflectance model," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 239–246.

[42] E. Heitz, J. Hanika, E. d'Eon, and C. Dachsbacher, "Multiple-scattering microfacet bsdfs with the smith model," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–14, 2016.

[43] S. Steinberg, P. Sen, and L.-Q. Yan, "Towards practical physical-optics rendering," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 1–13, 2022.

[44] E. d'Eon, B. Bitterli, A. Weidlich, and T. Zeltner, "Microfacet theory for non-uniform heightfields," in *ACM SIGGRAPH 2023 Conference Proceedings*, 2023, pp. 1–10.

[45] W. Jakob, M. Hašan, L.-Q. Yan, J. Lawrence, R. Ramamoorthi, and S. Marschner, "Discrete stochastic microfacet models," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–10, 2014.

[46] L.-Q. Yan, M. Hašan, S. Marschner, and R. Ramamoorthi, "Position-normal distributions for efficient rendering of specular microstructure," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–9, 2016.

[47] A. Ngan, F. Durand, and W. Matusik, "Experimental analysis of brdf models." *Rendering Techniques*, vol. 2005, no. 16th, p. 2, 2005.

[48] K. Zsolnai-Fehér, P. Wonka, and M. Wimmer, "Gaussian material synthesis," *arXiv preprint arXiv:1804.08369*, 2018.

[49] K. J. Dana, B. Van Ginneken, S. K. Nayar, and J. J. Koenderink, "Reflectance and texture of real-world surfaces," *ACM Transactions On Graphics (TOG)*, vol. 18, no. 1, pp. 1–34, 1999.

[50] A. Kuznetsov, "Neumip: Multi-resolution neural materials," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, 2021.

[51] C. Dostal and K. Yamafune, "Photogrammetric texture mapping: A method for increasing the fidelity of 3d models of cultural heritage materials," *Journal of Archaeological Science: Reports*, vol. 18, pp. 430–436, 2018.

[52] B. Burley and D. Lacewell, "Ptex: Per-face texture mapping for production rendering," in *Computer Graphics Forum*, vol. 27, no. 4. Wiley Online Library, 2008, pp. 1155–1164.

[53] J. Dong, J. Liu, K. Yao, M. Chantler, L. Qi, H. Yu, and M. Jian, "Survey of procedural methods for two-dimensional texture generation," *Sensors*, vol. 20, no. 4, p. 1135, 2020.

[54] J. Knodt, Z. Pan, K. Wu, and X. Gao, "Joint uv optimization and texture baking," *ACM Transactions on Graphics*, 2023.

[55] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Diversified texture synthesis with feed-forward networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3920–3928.

[56] L.-Y. Wie, S. Lefebvre, V. Kwatra, and G. Turk, "State of the Art in Example-based Texture Synthesis," in *Eurographics 2009 - State of the Art Reports*, M. Pauly and G. Greiner, Eds. The Eurographics Association, 2009.

[57] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2. IEEE, 1999, pp. 1033–1038.

[58] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 465–470.

[59] O. Wilson, A. Van Gelder, and J. Wilhelms, "Direct volume rendering via 3d textures," *Ucsc-crl-94-19, Jack Baskin School of Eng., University of California at Santa Cruz*, 1994.

[60] C. Yuksel, S. Lefebvre, and M. Tarini, "Rethinking texture mapping," in *Computer graphics forum*, vol. 38, no. 2. Wiley Online Library, 2019, pp. 535–551.

[61] J. P. Ewins, M. D. Waller, M. White, and P. F. Lister, "Mip-map level selection for texture mapping," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 4, pp. 317–329, 1998.

[62] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.

[63] P. G. Vivo and J. Lowe, "The book of shaders," *Dosegljivo: https://thebookofshaders. com*, 2015.

[64] S. Worley, "A cellular texture basis function," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 291–294.

[65] M. S. Mikkelsen, "Surface gradient–based bump mapping framework," *Journal of Computer Graphics Techniques Vol*, vol. 9, no. 3, 2020.

[66] V. Schüssler, E. Heitz, J. Hanika, and C. Dachsbacher, "Microfacet-based normal mapping for robust monte carlo path tracing," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 1–12, 2017.

[67] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, and S. Tachi, "Detailed shape representation with parallax mapping," in *Proceedings of ICAT*, vol. 2001, 2001, pp. 205–208.

[68] L. Szirmay-Kalos and T. Umenhoffer, "Displacement mapping on the gpu—state of the art," in *Computer graphics forum*, vol. 27, no. 6. Wiley Online Library, 2008, pp. 1567–1592.

[69] A. R. Smith, "Color gamut transform pairs," *ACM Siggraph Computer Graphics*, vol. 12, no. 3, pp. 12–19, 1978.

[70] I. E. S. of North America. Computer Committee, *IES standard file format for electronic transfer of photometric data and related information*. Illuminating Engineering Society of North America, 1991.

[71] N. Greene, "Environment mapping and other applications of world projections," *IEEE computer graphics and Applications*, vol. 6, no. 11, pp. 21–29, 1986.

[72] W. Heidrich, P. Slusallek, and H.-P. Seidel, "An image-based model for realistic lens systems in interactive computer graphics," in *Graphics Interface*, vol. 97. Citeseer, 1997, pp. 68–75.

[73] C. Kolb, D. Mitchell, and P. Hanrahan, "A realistic camera model for computer graphics," in *Proceedings of the 22nd annual conference on computer graphics and interactive techniques*, 1995, pp. 317–324.

[74] J. Wu, C. Zheng, X. Hu, and F. Xu, "Rendering realistic spectral bokeh due to lens stops and aberrations," *The Visual Computer*, vol. 29, pp. 41–52, 2013.

[75] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pp. 143–150.

[76] P. S. Heckbert, "Adaptive radiosity textures for bidirectional ray tracing," in *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990, pp. 145–154.

[77] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 213–222, 1984.

[78] A. Fujimoto, T. Tanaka, and K. Iwata, "Arts: Accelerated ray-tracing system," *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16–26, 1986.

[79] M. R. Kaplan, "The use of spatial coherence in ray tracing," in *ACM SIGGRAPH*, vol. 85, 1985, pp. 22–26.

[80] A. S. Glassner, "Space subdivision for fast ray tracing," *IEEE Computer Graphics and applications*, vol. 4, no. 10, pp. 15–24, 1984.

[81] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Communications of the ACM*, vol. 19, no. 10, pp. 547–554, 1976.

[82] I. Wald, "On fast construction of sah-based bounding volume hierarchies," in *2007 IEEE Symposium on Interactive Ray Tracing*. IEEE, 2007, pp. 33–40.

[83] T. Whitted, "An improved illumination model for shaded display," in *ACM Siggraph 2005 Courses*, 2005, pp. 4–es.

[84] M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S.-E. Yoon, "Recent advances in adaptive sampling and reconstruction for monte carlo rendering," in *Computer graphics forum*, vol. 34, no. 2. Wiley Online Library, 2015, pp. 667–681.

[85] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. X. Sillion, "A survey of real-time soft shadows algorithms," in *Computer Graphics Forum*, vol. 22, no. 4, 2003, pp. 753–774.

[86] M. Pharr and S. Green, "Ambient occlusion," *GPU Gems*, vol. 1, pp. 279–292, 2004.

[87] A. Silvennoinen and P.-P. Sloan, "Ray guiding for production lightmap baking," in *SIGGRAPH Asia 2019 Technical Briefs*, 2019, pp. 91–94.

[88] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel cone tracing," in *Computer Graphics Forum*, vol. 30, no. 7. Wiley Online Library, 2011, pp. 1921–1930.

[89] X. Yang, M. Yip, and X. Xu, "Visual effects in computer games," *Computer*, vol. 42, no. 7, pp. 48–56, 2009.

[90] R. Ganbar, *Nuke 101: professional compositing and visual effects*. Peachpit Press, 2014.

[91] A. Hertzmann, "Non-photorealistic rendering and the science of art," in *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, 2010, pp. 147–157.

[92] F. Durand and J. Dorsey, "Interactive tone mapping," in *Rendering Techniques 2000: Proceedings of the Eurographics Workshop in Brno, Czech Republic, June 26–28, 2000 11*. Springer, 2000, pp. 219–230.

[93] Y. Salih, A. S. Malik, N. Saad *et al.*, "Tone mapping of hdr images: A review," in *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, vol. 1. IEEE, 2012, pp. 368–373.

[94] H. S. Faridul, T. Pouli, C. Chamaret, J. Stauder, A. Trémeau, E. Reinhard *et al.*, "A survey of color mapping and its applications." *Eurographics (State of the Art Reports)*, vol. 3, no. 2, p. 1, 2014.

[95] K. D. A. C. A. Wilkie and W. Purgathofer, "Tone reproduction and physically based spectral rendering," in *Eurographics*, 2002.

[96] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes, "A survey on procedural modelling for virtual worlds," in *Computer Graphics Forum*, vol. 33, no. 6. Wiley Online Library, 2014, pp. 31–50.

[97] A. Raistrick, L. Lipson, Z. Ma, L. Mei, M. Wang, Y. Zuo, K. Kayan, H. Wen, B. Han, Y. Wang, A. Newell, H. Law, A. Goyal, K. Yang, and J. Deng, "Infinite photorealistic worlds using procedural generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2023, pp. 12 630–12 641.

[98] P. Guerrero, M. Hašan, K. Sunkavalli, R. Měch, T. Boubekeur, and N. J. Mitra, "Matformer: A generative model for procedural materials," *arXiv preprint arXiv:2207.01044*, 2022.

[99] A. Emilien, "Interactive design of virtual worlds: Combining procedural modeling with intuitive user control," Ph.D. dissertation, Université de Grenoble, 2014.

[100] S. Longay, A. Runions, F. Boudon, and P. Prusinkiewicz, "Treesketch: Interactive procedural modeling of trees on a tablet." in *SBIM@ Expressive*. Citeseer, 2012, pp. 107–120.

[101] Z. Hu and X. Qin, "Extended interactive and procedural modeling method for ancient chinese architecture," *Multimedia Tools and Applications*, vol. 80, pp. 5773–5807, 2021.

[102] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang, "Interactive procedural street modeling," in *ACM SIGGRAPH 2008 papers*, 2008, pp. 1–10.

[103] L. Krecklau and L. Kobbelt, "Interactive modeling by procedural high-level primitives," *Computers & Graphics*, vol. 36, no. 5, pp. 376–386, 2012.

[104] J. Freiknecht and W. Effelsberg, "A survey on the procedural generation of virtual worlds," *Multimodal Technologies and Interaction*, vol. 1, no. 4, p. 27, 2017.

[105] J. Freiknecht, "Procedural content generation for games," Ph.D. dissertation, Mannheim University, 2021.

[106] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *The ITB Journal*, vol. 7, no. 2, p. 5, 2006.

[107] SideFX, "Houdini." [Online]. Available: https://www.sidefx.com/

[108] C. Stover. Germ-grain model. [Online]. Available: https://mathworld.wolfram.com/Germ-GrainModel.html

[109] S. Glanville, "Texture bombing," *GPU Gems: Programming Techniques, Tips, and Tricks for*, vol. 1, 2004.

[110] "Domain warping," https://iquilezles.org/articles/warp/, accessed: 2023-08-03.

[111] S. Weiss, F. Bayer, and R. Westermann, "Triplanar displacement mapping for terrain rendering," *Eurographics 2020 - Short Papers*, 2020.

[112] A. Witkin and M. Kass, "Reaction-diffusion textures," in *Proceedings of the 18th annual conference on computer graphics and interactive techniques*, 1991, pp. 299–308.

[113] Y. Hu, C. He, V. Deschaintre, J. Dorsey, and H. Rushmeier, "An inverse procedural modeling pipeline for svbrdf maps," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 2, pp. 1–17, 2022.

[114] L. Shi, B. Li, M. Hašan, K. Sunkavalli, T. Boubekeur, R. Mech, and W. Matusik, "Match: Differentiable material graphs for procedural material capture," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–15, 2020.

[115] Y. Hu, P. Guerrero, M. Hasan, H. Rushmeier, and V. Deschaintre, "Node graph optimization using differentiable proxies," in *ACM SIGGRAPH 2022 conference proceedings*, 2022, pp. 1–9.

[116] I. Quilez and P. Jeremias, "Shadertoy," *Retrieved March*, vol. 27, p. 2017, 2017.

[117] G. Y. Gardner, "Visual simulation of clouds," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, 1985, pp. 297–304.

[118] C. Yuksel, S. Schaefer, and J. Keyser, "Parameterization and applications of catmull–rom curves," *Computer-Aided Design*, vol. 43, no. 7, pp. 747–755, 2011.

[119] M. Bailey and S. Cunningham, *Graphics shaders: theory and practice*. AK Peters/CRC Press, 2009.

[120] L. M. B. B. Fernandes, "Enhancing mesh deformation realism: Dynamic mesostructure detailing and procedural microstructure synthesis," Master's thesis, FEUP - Faculdade de Engenharia, 2023.

[121] K. Perlin, "Improving noise," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002, pp. 681–682.

[122] K. Perlin and E. M. Hoffert, "Hypertexture," in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, 1989, pp. 253–262.

[123] K. Perlin and F. Neyret, "Flow noise," in *28th International Conference on Computer Graphics and Interactive Techniques (Technical Sketches and Applications)*. SIGGRAPH, 2001, p. 187.

[124] J. J. Van Wijk, "Spot noise texture synthesis for data visualization," in *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 1991, pp. 309–318.

[125] A. Tsirikoglou, G. Eilertsen, and J. Unger, "A survey of image synthesis methods for visual machine learning," in *Computer Graphics Forum*, vol. 39, no. 6. Wiley Online Library, 2020, pp. 426–451.

[126] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4040–4048.

[127] A. Fabri and S. Pion, "Cgal: The computational geometry algorithms library," in *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2009, pp. 538–539.

[128] J. Møller and D. Stoyan, *Stochastic Geometry and Random Tessellations*, ser. Research Report Series. Department of Mathematical Sciences, Aalborg University, 2007, no. R-2007-28.

[129] B. Figliuzzi, "Introduction to stochastic geometry," Tech. Rep., 2015.[Online]. Available: http://www. cmm. mines-paristech. fr ..., Tech. Rep., 2015.

[130] S. C. Schvartzman and M. A. Otaduy, "Fracture animation based on high-dimensional voronoi diagrams," in *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2014, pp. 15–22.

[131] P. Prusinkiewicz and A. Lindenmayer, *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

[132] J. C. Hart, "The object instancing paradigm for linear fractal modeling," in *Graphics interface*, vol. 92. Citeseer, 1992, pp. 224–231.

[133] R. Ghosh and J. Marecek, "Iterated function systems: A comprehensive survey," *arXiv preprint arXiv:2211.14661*, 2022.

[134] B. B. Mandelbrot, "Fractal aspects of the iteration of z$\rightarrow$ λz (1-z) for complex λ and z," *Annals of the New York Academy of Sciences*, vol. 357, no. 1, pp. 249–259, 1980.

[135] T. A. Witten and L. M. Sander, "Diffusion-limited aggregation," *Physical review B*, vol. 27, no. 9, p. 5686, 1983.

[136] S. Havemann, "Generative mesh modeling," Ph.D. dissertation, Havemann, 2005.

[137] M. Mantyla and R. Sulonen, "Gwb: A solid modeler with euler operators," *IEEE Computer Graphics and Applications*, vol. 2, no. 7, pp. 17–31, 1982.

[138] S. Lesser, A. Stomakhin, G. Daviet, J. Wretborn, J. Edholm, N.-H. Lee, E. Schweickart, X. Zhai, S. Flynn, and A. Moffat, "Loki: a unified multiphysics simulation framework for production," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–20, 2022.

[139] D. G. Aliaga, İ. Demir, B. Benes, and M. Wand, "Inverse procedural modeling of 3d models for virtual worlds," in *ACM SIGGRAPH 2016 Courses*, 2016, pp. 1–316.

[140] J. Weissenberg, "Inverse procedural modelling and applications," Ph.D. dissertation, ETH-Zürich, 2014.

[141] M. Bokeloh, M. Wand, and H.-P. Seidel, "A connection between partial symmetry and inverse procedural modeling," in *ACM SIGGRAPH 2010 papers*, 2010, pp. 1–10.

[142] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes, "Inverse procedural modelling of trees," in *Computer Graphics Forum*, vol. 33, no. 6. Wiley Online Library, 2014, pp. 118–131.

[143] J. Guo, H. Jiang, B. Benes, O. Deussen, X. Zhang, D. Lischinski, and H. Huang, "Inverse procedural modeling of branching structures by inferring l-systems," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 5, pp. 1–13, 2020.

[144] F. Wu, D.-M. Yan, W. Dong, X. Zhang, and P. Wonka, "Inverse procedural modeling of facade layouts," *arXiv preprint arXiv:1308.0419*, 2013.

[145] I. Demir, D. G. Aliaga, and B. Benes, "Proceduralization for editing 3d architectural models," in *2016 Fourth International Conference on 3D Vision (3DV)*. IEEE, 2016, pp. 194–202.

[146] O. Pearl, I. Lang, Y. Hu, R. A. Yeh, and R. Hanocka, "Geocode: Interpretable shape programs," *arXiv preprint arXiv:2212.11715*, 2022.

[147] M. Shechter, R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, "Neuralmls: Geometry-aware control point deformation," *arXiv preprint arXiv:2201.01873*, 2022.

[148] R. L. Cook, "Shade trees," in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984, pp. 223–231.

[149] W. Alan and W. Mark, "Advanced animation and rendering techniques," *Theory and Practice Wokingham*, pp. 339–368, 1992.

[150] Blender.Foundation, "Blender." [Online]. Available: https://www.blender.org/

[151] M. E. Mortenson, *Geometric modeling*. John Wiley & Sons, Inc., 1997.

[152] J. Gomes, *Warping & morphing of graphical objects*. Morgan Kaufmann, 1999.

[153] J. Portsmouth, "Efficient barycentric point sampling on meshes," *arXiv preprint arXiv:1708.07559*, 2017.

[154] M. Šik and J. Křivánek, "Fast random sampling of triangular meshes," in *Pacific Graphics, Short Papers*, 2013.

[155] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe, "Fast exact and approximate geodesics on meshes," *ACM transactions on graphics (TOG)*, vol. 24, no. 3, pp. 553–560, 2005.

[156] C. M. Hoffmann, *Geometric and solid modeling: an introduction*. Morgan Kaufmann Publishers Inc., 1989.

[157] D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes, "Constructive solid geometry for polyhedral objects," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pp. 161–170.

[158] D. Khan, A. Plopski, Y. Fujimoto, M. Kanbara, G. Jabeen, Y. J. Zhang, X. Zhang, and H. Kato, "Surface remeshing: A systematic literature review of methods and research directions," *IEEE transactions on visualization and computer graphics*, vol. 28, no. 3, pp. 1680–1713, 2020.

[159] R. M. Smelik, K. J. De Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen, "A survey of procedural methods for terrain modelling," in *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, vol. 2009. sn, 2009, pp. 25–34.

[160] A. Runions, B. Lane, and P. Prusinkiewicz, "Modeling trees with a space colonization algorithm." *Nph*, vol. 7, no. 63–70, p. 6, 2007.

[161] J.-D. Génevaux, É. Galin, E. Guérin, A. Peytavie, and B. Benes, "Terrain generation using procedural models based on hydrology," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–13, 2013.

[162] F. Belhadj and P. Audibert, "Modeling landscapes with ridges and rivers: bottom up approach," in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 2005, pp. 447–450.

[163] Y. I. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 301–308.

[164] E-ON Software, "VUE." [Online]. Available: https://info.e-onsoftware.com/vue/overview

[165] Planetside Software, "Terragen." [Online]. Available: https://planetside.co.uk/

[166] Unity Technologies, "Speed Tree." [Online]. Available: https://store.speedtree.com/

[167] Xfrog inc., "Xfrog." [Online]. Available: https://www.xfrog.net/

[168] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, 12 2019.

[169] W. Tang, Q. Yang, K. Xiong, and W. Yan, "Deep learning based automatic defect identification of photovoltaic module using electroluminescence images," *Solar Energy*, vol. 201, pp. 453 – 460, 2020.

[170] S. Mérillou, J.-M. Dischler, and D. Ghazanfarpour, "Surface scratches: measuring, modeling and rendering," *The Visual Computer*, vol. 17, no. 1, pp. 30–45, 2001.

[171] B. Desbenoit, E. Galin, and S. Akkouche, "Modeling cracks and fractures," *The Visual Computer*, vol. 21, pp. 717–726, 2005.

[172] M. Haindl and J. Filip, *Visual texture: Accurate material appearance measurement, representation and modeling.* Springer Science & Business Media, 2013.

[173] J. Reiner, "Rendering for machine vision prototyping," in *Optical Design and Engineering III*, vol. 7100. International Society for Optics and Photonics, 2008, p. 710009.

[174] Z. Dong, B. Walter, S. Marschner, and D. P. Greenberg, "Predicting appearance from measured microgeometry of metal surfaces," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 1, pp. 1–13, 2015.

[175] H. Kolivand, M. S. Sunar, S. Y. Kakh, R. Al-Rousan, and I. Ismail, "Photorealistic rendering: a survey on evaluation," *Multimedia Tools and Applications*, vol. 77, no. 19, pp. 25 983–26 008, 2018.

[176] D. P. Greenberg, K. E. Torrance, P. Shirley, J. Arvo, E. Lafortune, J. A. Ferwerda, B. Walter, B. Trumbore, S. Pattanaik, and S.-C. Foo, "A framework for realistic image synthesis," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 477–494.

[177] J. D. Stets, A. D. Corso, J. B. Nielsen, R. A. Lyngby, S. H. N. Jensen, J. Wilm, M. B. Doest, C. Gundlach, E. R. Eiriksson, K. Conradsen, A. B. Dahl, J. A. Bærentzen, J. R. Frisvad, and H. Aanæs, "Scene reassembly after multimodal digitization and pipeline evaluation using photorealistic rendering," *Appl. Opt.*, vol. 56, no. 27, pp. 7679–7690, Sep 2017. [Online]. Available: http://ao.osa.org/abstract.cfm?URI=ao-56-27-7679

[178] R. Y. Tsai and R. K. Lenz, "A new technique for fully autonomous and efficient 3d robotics hand/eye calibration," *IEEE Transactions on robotics and automation*, vol. 5, no. 3, pp. 345–358, 1989.

[179] K. Pachtrachai, M. Allan, V. Pawar, S. Hailes, and D. Stoyanov, "Hand-eye calibration for robotic assisted minimally invasive surgery without a calibration object," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2485–2491.

[180] M. Antonello, A. Gobbi, S. Michieletto, S. Ghidoni, and E. Menegatti, "A fully automatic hand-eye calibration system," in *2017 European Conference on Mobile Robots (ECMR)*, 2017, pp. 1–6.

[181] H. Wang, X. Lu, Z. Hu, and Y. Li, "A vision-based fully-automatic calibration method for hand-eye serial robot," *Industrial Robot: An International Journal*, 2015.

[182] R. A. Lyngby, J. B. Matthiassen, J. R. Frisvad, A. B. Dahl, and H. Aanæs, "Using a robotic arm for measuring brdfs," in *Scandinavian Conference on Image Analysis*. Springer, 2019, pp. 184–196.

[183] R. Liang and J. Mao, "Hand-eye calibration with a new linear decomposition algorithm," *Journal of Zhejiang University-SCIENCE A*, vol. 9, no. 10, pp. 1363–1368, 2008.

[184] F. Beaune, E. Tovagliari, L. Barrancos, S. Agyemang, S. Basu, M. Bhutani, L. Bosnar, R. Brune, M. Chan, J. M. M. Costa, H. Crepaz, J. Deng, J. Dent, M. Dhiman, D. Fevrier, K. R. Iyer, D. Lahiri, K. Masson, G. Olson, A. Pandey, J. Park, S. Pogosyan, B. Samir, O. Smolin, T. Vergne, L. Wilimitis, and L. Zawallich, "appleseed," Sep. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3456967

[185] L. Gritz, C. Stein, C. Kulla, and A. Conty, "Open shading language," in *ACM SIGGRAPH 2010 Talks*, 2010, pp. 1–1.

[186] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[187] É. Marchand, F. Spindler, and F. Chaumette, "Visp for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robotics & Automation Magazine*, vol. 12, no. 4, pp. 40–52, 2005.

[188] P. Gospodnetic, M. Rauhut, and H. Hagen, "Surface inspection planning using 3d visualization," *LEVIA*, 2020.

[189] P. Gospodnetic, D. Mosbach, M. Rauhut, and H. Hagen, "Viewpoint placement for inspection planning," *Machine Vision and Applications*, 2021.

[190] D. Mosbach, P. Gospodnetić, M. Rauhut, B. Hamann, and H. Hagen, "Feature-driven viewpoint placement for model-based surface inspection," *Machine Vision and Applications*, vol. 32, no. 1, pp. 1–21, 2020.

[191] M. Mohammadikaji, S. Bergmann, J. Beyerer, J. Burke, and C. Dachsbacher, "Sensor-realistic simulations for evaluation and planning of optical measurement systems with an application to laser triangulation," *IEEE Sensors Journal*, vol. 20, no. 10, pp. 5336–5349, 2020.

[192] M. Mohammadikaji, "Simulation-based planning of machine vision inspection systems with an application to laser triangulation," Ph.D. dissertation, KIT, 2020.

[193] T. B. Jørgensen, T. M. Iversen, A. P. Lindvig, C. Schlette, D. Kraft, T. R. Savarimuthu, J. Roßmann, and N. Krüger, "Simulation-based optimization of camera placement in the context of industrial pose estimation." in *VISIGRAPP (5: VISAPP)*, 2018, pp. 524–533.

[194] R. A. Hall and D. P. Greenberg, "A testbed for realistic image synthesis," *IEEE Computer Graphics and Applications*, vol. 3, no. 8, pp. 10–20, 1983.

[195] P. Dutre, P. Bekaert, and K. Bala, *Advanced global illumination*. CRC Press, 2018.

[196] Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or, and H. Huang, "Non-stationary texture synthesis by adversarial expansion," *arXiv preprint arXiv:1805.04487*, 2018.

[197] A. Frühstück, I. Alhashim, and P. Wonka, "Tilegan: synthesis of large-scale non-homogeneous textures," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–11, 2019.

[198] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker, "A survey of procedural noise functions," in *Computer Graphics Forum*, vol. 29, no. 8. Wiley Online Library, 2010, pp. 2579–2600.

[199] P. Vangorp, J. Laurijssen, and P. Dutré, "The influence of shape on the perception of material reflectance," in *ACM SIGGRAPH 2007 papers*, 2007, pp. 77–es.

[200] C. Bosch, "Realistic image synthesis of surface scratches and grooves," Ph.D. dissertation, Citeseer, 2007.

[201] P. Poulin and A. Fournier, "A model for anisotropic reflection," *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 4, pp. 273–282, 1990.

[202] B. Raymond, G. Guennebaud, P. Barla, R. Pacanowski, and X. Granier, "Optimizing brdf orientations for the manipulation of anisotropic highlights," in *Computer Graphics Forum*, vol. 33, no. 2. Wiley Online Library, 2014, pp. 313–321.

[203] C. Bosch, X. Pueyo, S. Mérillou, and D. Ghazanfarpour, "A physically-based model for rendering realistic scratches," in *Computer Graphics Forum*, vol. 23, no. 3. Wiley Online Library, 2004, pp. 361–370.

[204] ——, "General rendering of grooved surfaces," Citeseer, Tech. Rep., 2005.

[205] C. Bosch and G. Patow, "Real time scratches and grooves," in *XVII Congreso Espanol de Informática Gráfica (CEIG'07)*. Citeseer, 2007.

[206] C. Bosch, X. Pueyo, S. Mérillou, and D. Ghazanfarpour, "A resolution independent approach for the accurate rendering of grooved surfaces," in *Computer Graphics Forum*, vol. 27, no. 7. Wiley Online Library, 2008, pp. 1937–1944.

[207] B. Raymond, G. Guennebaud, and P. Barla, "Multi-scale rendering of scratched materials using a structured sv-brdf model," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–11, 2016.

[208] M. Mannan, Z. Mian, and A. A. Kassim, "Tool wear monitoring using a fast hough transform of images of machined surfaces," *Machine Vision and Applications*, vol. 15, no. 3, pp. 156–163, 2004.

[209] X. Jiang, P. J. Scott, D. Whitehouse, and L. Blunt, "Paradigm shifts in surface metrology. part i. historical philosophy," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 463, no. 2085, pp. 2049–2070, 2007.

[210] X. Jiang, P. J. Scott, D. J. Whitehouse, and L. Blunt, "Paradigm shifts in surface metrology. part ii. the current shift," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 463, no. 2085, pp. 2071–2099, 2007.

[211] "Iso 1302 (2002) geometrical product specifications (gps)—indication of surface texture in technical product documentation."

[212] G. W. Wolf, "Surfaces—topography and topology," *Surface Topography: Metrology and Properties*, vol. 8, no. 1, p. 014003, 2020.

[213] A. Humeau-Heurtier, "Texture feature extraction methods: A survey," *IEEE Access*, vol. 7, pp. 8975–9000, 2019.

[214] S. I. Nikolenko, "Synthetic data for deep learning," 2019. [Online]. Available: https://arxiv.org/abs/1909.11512

[215] P. Gutierrez, M. Luschkova, A. Cordier, M. Shukor, M. Schappert, and T. Dahmen, "Synthetic training data generation for deep learning based quality inspection," in *Fifteenth International Conference on Quality Control by Artificial Vision*, vol. 11794. SPIE, 2021, pp. 9–16.

[216] Q. Zhou, E. Grinspun, D. Zorin, and A. Jacobson, "Mesh arrangements for solid geometry," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–15, 2016.

[217] H. E. Rushmeier, B. E. Rogowitz, and C. Piatko, "Perceptual issues in substituting texture for geometry," in *Human Vision and Electronic Imaging V*, vol. 3959. Spie, 2000, pp. 372–383.

[218] O. Schmedemann, M. Baaß, D. Schoepflin, and T. Schüppstuhl, "Procedural synthetic training data generation for ai-based defect detection in industrial surface inspection," *Procedia CIRP*, vol. 107, pp. 1101–1106, 2022, leading manufacturing systems transformation – Proceedings of the 55th CIRP Conference on Manufacturing Systems 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827122003997

[219] A. Boikov, V. Payor, R. Savelev, and A. Kolesnikov, "Synthetic data generation for steel defect detection and classification using deep learning," *Symmetry*, vol. 13, no. 7, p. 1176, 2021.

[220] C. Bosch and G. Patow, "Real-time path-based surface detail," *Computers & Graphics*, vol. 34, no. 4, pp. 430–440, 2010.

[221] S. Niu, B. Li, X. Wang, and H. Lin, "Defect image sample generation with gan for improving defect recognition," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1611–1622, 2020.

[222] T. Czimmermann, G. Ciuti, M. Milazzo, M. Chiurazzi, S. Roccella, C. M. Oddo, and P. Dario, "Visual-based defect detection and classification approaches for industrial applications—a survey," *Sensors*, vol. 20, no. 5, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/5/1459

[223] S. Moonen, B. Vanherle, J. de Hoog, T. Bourgana, A. Bey-Temsamani, and N. Michiels, "CAD2Render: A modular toolkit for GPU-accelerated photorealistic synthetic data generation for the manufacturing industry," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, January 2023, pp. 583–592.

[224] T. Bao, J. Chen, W. Li, X. Wang, J. Fei, L. Wu, R. Zhao, and Y. Zheng, "Miad: A maintenance inspection dataset for unsupervised anomaly detection," 2022. [Online]. Available: https://arxiv.org/abs/2211.13968

[225] X. Guo, W. Wu, D. Wang, J. Su, H. Su, W. Gan, J. Huang, and Q. Yang, "Learning video representations of human motion from synthetic data," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 20 165–20 175.

[226] E. Wood, T. Baltrušaitis, C. Hewitt, S. Dziadzio, T. J. Cashman, and J. Shotton, "Fake it till you make it: Face analysis in the wild using synthetic data alone," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 3681–3691.

[227] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2018, pp. 1082–10 828. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPRW.2018.00143

[228] B. Tang, L. Chen, W. Sun, and Z.-k. Lin, "Review of surface defect detection of steel products based on machine vision," *IET Image Processing*, vol. 17, no. 2, pp. 303–322, 2023. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ipr2.12647

[229] X. Wen, J. Shan, Y. He, and K. Song, "Steel surface defect recognition: A survey," *Coatings*, vol. 13, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2079-6412/13/1/17

[230] J. Yang, S. Li, Z. Wang, H. Dong, J. Wang, and S. Tang, "Using deep learning to detect defects in manufacturing: A comprehensive survey and current challenges," *Materials*, vol. 13, no. 24, 2020. [Online]. Available: https://www.mdpi.com/1996-1944/13/24/5755

[231] T. Barisin, C. Jung, F. Müsebeck, C. Redenbach, and K. Schladitz, "Methods for segmenting cracks in 3d images of concrete: A comparison based on semi-synthetic images," *Pattern Recognition*, vol. 129, p. 108747, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S003132032200228X

[232] Q. Fang, C. Ibarra-Castanedo, and X. Maldague, "Automatic defects segmentation and identification by deep learning algorithm with pulsed thermography: Synthetic and experimental data," *Big Data and Cognitive Computing*, vol. 5, no. 1, 2021. [Online]. Available: https://www.mdpi.com/2504-2289/5/1/9

[233] Y. Chen, Y. Ding, F. Zhao, E. Zhang, Z. Wu, and L. Shao, "Surface defect detection methods for industrial products: A review," *Applied Sciences*, vol. 11, no. 16, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/16/7657

[234] A.-A. Tulbure, A.-A. Tulbure, and E.-H. Dulf, "A review on modern defect detection models using dcnns – deep convolutional neural networks," *Journal of Advanced Research*, vol. 35, pp. 33–48, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090123221000643

[235] D. Tabernik, S. Šela, J. Skvarč, and D. Skočaj, "Segmentation-Based Deep-Learning Approach for Surface-Defect Detection," *Journal of Intelligent Manufacturing*, May 2019.

[236] J. Božič, D. Tabernik, and D. Skočaj, "Mixed supervision for surface-defect detection: from weakly to fully supervised learning," *Computers in Industry*, 2021.

[237] D. Honzátko, E. Türetken, S. A. Bigdeli, L. A. Dunbar, and P. Fua, "Defect segmentation for multi-illumination quality control systems,"

Machine Vision and Applications, vol. 32, no. 6, p. 118, Sep 2021. [Online]. Available: https://doi.org/10.1007/s00138-021-01244-z

[238] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[239] H. Lin, B. Li, X. Wang, Y. Shu, and S. Niu, "Automated defect inspection of led chip using deep convolutional neural network," *Journal of Intelligent Manufacturing*, vol. 30, pp. 1–10, 08 2019.

[240] M. Rudolph, T. Wehrbein, B. Rosenhahn, and B. Wandt, "Asymmetric student-teacher networks for industrial anomaly detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2023, pp. 2592–2602.

[241] J. Liu, G. Xie, J. Wang, S. Li, C. Wang, F. Zheng, and Y. Jin, "Deep industrial image anomaly detection: A survey," 2023. [Online]. Available: https://arxiv.org/abs/2301.11514

[242] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, and P. Gehler, "Towards total recall in industrial anomaly detection," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 14 298–14 308.

[243] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger, "Mvtec ad — a comprehensive real-world dataset for unsupervised anomaly detection," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9584–9592.

[244] X. Sun, J. Gu, S. Tang, and J. Li, "Research progress of visual inspection technology of steel products—a review," *Applied Sciences*, vol. 8, no. 11, 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/11/2195

[245] P. Severstal, "Severstal: Steel defect detection," https://www.kaggle.com/c/severstal-steel-defect-detection, 2019.

[246] K. Song and Y. Yan, "A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects," *Applied Surface Science*, vol. 285, pp. 858–864, 11 2013. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0169433213016437

[247] G. Fu, P. Sun, W. Zhu, J. Yang, Y. Cao, M. Y. Yang, and Y. Cao, "A deep-learning-based approach for fast and robust steel surface

defects classification," *Optics and Lasers in Engineering*, vol. 121, pp. 397–405, 10 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0143816619301678

[248] X. Lv, F. Duan, J.-j. Jiang, X. Fu, and L. Gan, "Deep metallic surface defect detection: The new benchmark and detection network," *Sensors*, vol. 20, no. 6, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/6/1562

[249] Y. Huang, C. Qiu, Y. Guo, X. Wang, and K. Yuan, "Surface defect saliency of magnetic tile," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 2018, pp. 612–617.

[250] Tianchi, "Aluminum profile surface flaw recognition dataset," 2016. [Online]. Available: https://tianchi.aliyun.com/dataset/dataDetail?dataId=140666

[251] Z. Zhang, S. Yu, S. Yang, Y. Zhou, and B. Zhao, "Rail-5k: a real-world dataset for rail surface defects detection [unpublished]," *CoRR*, vol. abs/2106.14366, 2021. [Online]. Available: https://arxiv.org/abs/2106.14366

[252] T. Schlagenhauf, M. Landwehr, and J. Fleischer, "Industrial machine tool element surface defect dataset," 2021.

[253] P. Mishra, R. Verk, D. Fornasier, C. Piciarelli, and G. L. Foresti, "VT-ADL: A vision transformer network for image anomaly detection and localization," in *30th IEEE/IES International Symposium on Industrial Electronics (ISIE)*, June 2021.

[254] M. Haselmann and D. Gruber, "Supervised machine learning based surface inspection by synthetizing artificial defects," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, pp. 390–395.

[255] G. Zhang, K. Cui, T.-Y. Hung, and S. Lu, "Defect-GAN: High-fidelity defect synthesis for automated defect inspection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021, pp. 2524–2534.

[256] R. Wang, S. Hoppe, E. Monari, and M. Huber, "Defect transfer gan: Diverse defect synthesis for data augmentation," in *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022. [Online]. Available: https://bmvc2022.mpi-inf.mpg.de/0445.pdf

[257] T. Dahmen, P. Trampert, F. Boughorbel, J. Sprenger, M. Klusch, K. Fischer, C. Kübel, and P. Slusallek, "Digital reality: a model-based approach to supervised learning from synthetic data," *AI Perspectives*, vol. 1, no. 1, p. Article No.2, 2019, 43.22.02; LK 01.

[258] M. Wieler, T. Hahn, and F. A. Hamprecht, "Weakly supervised learning for industrial optical inspection [dataset]," https://hci.iwr.uni-heidelberg.de/content/weakly-supervised-learning-industrial-optical-inspection, 2007.

[259] A. Boikov, V. Payor, R. Savelev, and A. Kolesnikov, "Synthetic data generation for steel defect detection and classification using deep learning," *Symmetry*, vol. 13, no. 7, 2021. [Online]. Available: https://www.mdpi.com/2073-8994/13/7/1176

[260] P. De Roovere, S. Moonen, N. Michiels, and F. Wyffels, "Dataset of industrial metal objects," 2022. [Online]. Available: https://arxiv.org/abs/2208.04052

[261] G. Csurka, "Domain adaptation for visual applications: A comprehensive survey," 2017. [Online]. Available: https://arxiv.org/abs/1702.05374

[262] C. Wu, X. Bi, J. Pfrommer, A. Cebulla, S. Mangold, and J. Beyerer, "Sim2real transfer learning for point cloud segmentation: An industrial application case on autonomous disassembly," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2023, pp. 4531–4540.

[263] B. Vanherle, S. Moonen, F. V. Reeth, and N. Michiels, "Analysis of training object detection models with synthetic data," in *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022. [Online]. Available: https://bmvc2022.mpi-inf.mpg.de/0833.pdf

[264] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

[265] K. Wada, "Labelme: Image polygonal annotation with Python." [Online]. Available: https://github.com/wkentaro/labelme

[266] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2015, pp. 3431–3440. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298965

[267] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," *arXiv e-prints*, p. arXiv:1706.05587, Jun. 2017.

[268] O. Ronneberger, P.Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351. Springer, 2015, pp. 234–241, (available on arXiv:1505.04597 [cs.CV]). [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a

[269] P. Iakubovskii, "Segmentation models," https://github.com/qubvel/segmentation_models, 2019.

[270] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[271] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2017.

[272] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[273] X. Wu, K. Xu, and P. Hall, "A survey of image synthesis and editing with generative adversarial networks," *Tsinghua Science and Technology*, vol. 22, no. 6, pp. 660–674, 2017.

[274] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, "A review on generative adversarial networks: Algorithms, theory, and applications," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 4, pp. 3313–3332, 2021.

[275] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (gans): A survey," *IEEE access*, vol. 7, pp. 36 322–36 333, 2019.

[276] A. Figueira and B. Vaz, "Survey on synthetic data generation, evaluation methods and GANs," *Mathematics*, vol. 10, no. 15, 2022.

[277] C. Little, M. Elliot, R. Allmendinger, and S. S. Samani, "Generative adversarial networks for synthetic data generation: a comparative study," *arXiv preprint arXiv:2112.01925*, 2021.

[278] J. T. Guibas, T. S. Virdi, and P. S. Li, "Synthetic medical images from dual generative adversarial networks," *arXiv preprint arXiv:1709.01872*, 2017.

[279] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy, "Esrgan: Enhanced super-resolution generative adversarial networks," in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[280] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[281] U. Bergmann, N. Jetchev, and R. Vollgraf, "Learning texture manifolds with the periodic spatial gan," *arXiv preprint arXiv:1705.06566*, 2017.

[282] R. Huang, S. Zhang, T. Li, and R. He, "Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2439–2448.

[283] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2536–2544.

[284] B. Zhao, X. Wu, Z.-Q. Cheng, H. Liu, Z. Jie, and J. Feng, "Multi-view image generation from a single-view," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 383–391.

[285] L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool, "Pose guided person image generation," in *Proceedings of 31st Conference on Neural Information Processing Systems*, vol. 30, 2017.

[286] M. Haselmann and D. P. Gruber, "Pixel-wise defect detection by cnns without manually labeled training data," *Applied Artificial Intelligence*, vol. 33, no. 6, pp. 548–566, 2019. [Online]. Available: https://doi.org/10.1080/08839514.2019.1583862

[287] O. Schmedemann, S. Schlodinski, D. Holst, and T. Schüppstuhl, "Adapting synthetic training data in deep learning-based visual surface inspection to improve transferability of simulations to real-world environments," in *Automated Visual Inspection and Machine Vision V*, vol. 12623. SPIE, 2023, pp. 25–35.

[288] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, "Learning from synthetic humans," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 109–117.

[289] D. Park and D. Ramanan, "Articulated pose estimation with tiny synthetic videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 58–66.

[290] M. Fabbri, F. Lanzi, S. Calderara, A. Palazzi, R. Vezzani, and R. Cucchiara, "Learning to detect and track visible and occluded body joints in a virtual world," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 430–446.

[291] A. Kortylewski, A. Schneider, T. Gerig, B. Egger, A. Morel-Forster, and T. Vetter, "Training deep face recognition systems with synthetic data," *arXiv preprint arXiv:1802.05891*, 2018.

[292] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[293] A. Shafaei, J. J. Little, and M. Schmidt, "Play and learn: Using video games to train computer vision models," *arXiv preprint arXiv:1608.01745*, 2016.

[294] B. Hurl, K. Czarnecki, and S. Waslander, "Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 2522–2529.

[295] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," *arXiv preprint arXiv:1809.10790*, 2018.

[296] "Nvidia isaac - the accelerated platform for robotics and ai," https://www.nvidia.com/en-us/deep-learning-ai/industries/robotics/, accessed: 2023-01-30.

[297] S. Khan, B. Phan, R. Salay, and K. Czarnecki, "Procsy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks." in *CVPR workshops*, vol. 3, 2019, p. 4.

[298] E. Buls, R. Kadikis, R. Cacurs, and J. Ārents, "Generation of synthetic training data for object detection in piles," in *Eleventh International Conference on Machine Vision (ICMV 2018)*, vol. 11041. SPIE, 2019, pp. 533–540.

[299] M. Wieler and T. Hahn, "Weakly supervised learning for industrial optical inspection," in *DAGM symposium in*, vol. 6, 2007.

[300] T. Bao, J. Chen, W. Li, X. Wang, J. Fei, L. Wu, R. Zhao, and Y. Zheng, "Miad: A maintenance inspection dataset for unsupervised

anomaly detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 993–1002.

[301] F. A. Saiz, G. Alfaro, I. Barandiaran, S. Garcia, M. Carretero, and M. Graña, "Synthetic data set generation for the evaluation of image acquisition strategies applied to deep learning based industrial component inspection systems." in *CEIG*, 2021, pp. 1–8.

[302] B. Yang, Z. Liu, G. Duan, and J. Tan, "Mask2defect: A prior knowledge-based data augmentation method for metal surface defect inspection," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 6743–6755, 2021.

[303] A. G. Abubakr, I. Jovancevic, N. I. Mokhtari, H. B. Abdallah, and J.-J. Orteu, "On learning deep domain-invariant features from 2d synthetic images for industrial visual inspection," in *Fifteenth International Conference on Quality Control by Artificial Vision*, vol. 11794. SPIE, 2021, pp. 317–325.

[304] A. Kim, K. Lee, S. Lee, J. Song, S. Kwon, and S. Chung, "Synthetic data and computer-vision-based automated quality inspection system for reused scaffolding," *Applied Sciences*, vol. 12, no. 19, p. 10097, 2022.

[305] S. Sauer, M. Borkar, D. Sasidharan, and T. Dunker, "Model-based visual inspection with machine learning methods using simulation of the expected camera view," Workshop on "Generating synthetic image data for AI" at the KI 2022 (virtual, hosted in Trier/Germany), September 2022.

[306] K. Greff, F. Belletti, L. Beyer, C. Doersch, Y. Du, D. Duckworth, D. J. Fleet, D. Gnanapragasam, F. Golemo, C. Herrmann *et al.*, "Kubric: A scalable dataset generator," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3749–3761.

[307] S. Hill, S. McAuley, L. Belcour, W. Earl, N. Harrysson, S. Hillaire, N. Hoffman, L. Kerley, J. Patry, R. Pieké *et al.*, "Physically based shading in theory and practice," in *ACM SIGGRAPH 2020 Courses*, 2020, pp. 1–12.

[308] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, jan 1949. [Online]. Available: https://doi.org/10.1109/jrproc.1949.232969

[309] C. Kulla and A. Conty, "Revisiting physically based shading at imageworks," *SIGGRAPH Course, Physically Based Shading*, vol. 2, no. 3, 2017.

[310] E. Turquin, "Practical multiple scattering compensation for microfacet models," *URL: https://blog. selfshadow. com/publications/turquin/ms_ comp_final. pdf*, vol. 45, no. 3, 2019.

[311] "The pbr guide by allegorithmic - part 2," https://substance3d.adobe.com/tutorials/courses/the-pbr-guide-part-2, accessed: 2023-01-30.

[312] M. S. Mikkelsen, "Bump mapping unparametrized surfaces on the gpu," *Journal of graphics, gpu, and game tools*, vol. 15, no. 1, pp. 49–61, 2010.

[313] "Freecad - an open-source parametric 3d modeler," https://www.freecadweb.org/, accessed: 2023-01-30.

[314] T. Duff, J. Burgess, P. Christensen, C. Hery, A. Kensler, M. Liani, and R. Villemin, "Building an orthonormal basis, revisited," *JCGT*, vol. 6, no. 1, 2017.

[315] J. Zhu, S. Zhao, Y. Xu, X. Meng, L. Wang, and L.-Q. Yan, "Recent advances in glinty appearance rendering," *Computational Visual Media*, vol. 8, no. 4, pp. 535–552, 2022.

[316] X. Chermain, F. Claux, and S. Mérillou, "Glint rendering based on a multiple-scattering patch brdf," in *Computer Graphics Forum*, vol. 38, no. 4. Wiley Online Library, 2019, pp. 27–37.

[317] M. P. Groover, *Fundamentals of modern manufacturing: materials, processes, and systems*. John Wiley & Sons, 2020.

[318] T. H. Childs, K. Maekawa, T. Obikawa, and Y. Yamane, *Metal machining: theory and applications*. Butterworth-Heinemann, 2000.

[319] C. Felhő, B. Karpuschewski, and J. Kundrák, "Surface roughness modelling in face milling," *Procedia CIRP*, vol. 31, pp. 136–141, 2015.

[320] C. Felhő and J. Kundrák, "Effects of setting errors (insert run-outs) on surface roughness in face milling when using circular inserts," *Machines*, vol. 6, no. 2, 2018.

[321] J. Kundrák, C. Felhő, and A. Nagy, "Analysis and prediction of roughness of face milled surfaces using cad model," *Manufacturing Technology*, vol. 22, pp. 558–572, 2022.

[322] M. Hadad and M. Ramezani, "Modeling and analysis of a novel approach in machining and structuring of flat surfaces using face milling process," *International Journal of Machine Tools and Manufacture*, vol. 105, pp. 32–44, 2016.

[323] M. Ritz, S. Breitfelder, P. Santos, A. Kuijper, and D. W. Fellner, "Seamless and non-repetitive 4d texture variation synthesis and real-time rendering for measured optical material behavior," *Computational Visual Media*, vol. 5, pp. 161–170, 2019.

[324] N. Jetchev, U. Bergmann, and R. Vollgraf, "Texture synthesis with spatial generative adversarial networks," *arXiv preprint arXiv:1611.08207*, 2016.

[325] T. Zeltner, F. Rousselle, A. Weidlich, P. Clarberg, J. Novák, B. Bitterli, A. Evans, T. Davidovič, S. Kallweit, and A. Lefohn, "Real-time neural appearance models," *arXiv preprint arXiv:2305.02678*, 2023.

[326] L. Raad, A. Davy, A. Desolneux, and J.-M. Morel, "A survey of exemplar-based texture synthesis," 2017.

[327] B. Galerne, Y. Gousseau, and J.-M. Morel, "Random phase textures: Theory and synthesis," *IEEE Transactions on Image Processing*, vol. 20, no. 1, pp. 257–267, 2011.

[328] D. Heeger and J. Bergen, "Pyramid-based texture analysis/synthesis," in *Proceedings., International Conference on Image Processing*, vol. 3, 1995, pp. 648–651 vol.3.

[329] T. Briand, J. Vacher, B. Galerne, and J. Rabin, "The Heeger & Bergen Pyramid Based Texture Synthesis Algorithm," *Image Processing On Line*, vol. 4, pp. 276–299, 2014, https://doi.org/10.5201/ipol.2014.79.

[330] J. Portilla and E. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *International Journal of Computer Vision*, vol. 40, 10 2000.

[331] J. Vacher and T. Briand, "The portilla-simoncelli texture model: towards understanding the early visual cortex," *Image Processing On Line*, vol. 11, pp. 170–211, 2021, https://doi.org/10.5201/ipol.2021.324.

[332] A. Efros and W. Freeman, "Image quilting for texture synthesis and transfer," *Computer Graphics (Proc. SIGGRAPH'01)*, vol. 35, 07 2001.

[333] L. Raad and B. Galerne, "Efros and Freeman Image Quilting Algorithm for Texture Synthesis," *Image Processing On Line*, vol. 7, pp. 1–22, 2017, https://doi.org/10.5201/ipol.2017.171.

[334] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1033–1038.

[335] P. Guehl, R. Allegre, J.-M. Dischler, B. Benes, and E. Galin, "Semi-procedural textures using point process texture basis functions," in *Computer Graphics Forum*, vol. 39, no. 4. Wiley Online Library, 2020, pp. 159–171.

[336] C. Jung and C. Redenbach, "Crack modeling via minimum-weight surfaces in 3d voronoi diagrams," 2022.

[337] D. Tabernik, S. Šela, J. Skvarč, and D. Skočaj, "Segmentation-based deep-learning approach for surface-defect detection," *Journal of Intelligent Manufacturing*, vol. 31, no. 3, pp. 759–776, Mar 2020. [Online]. Available: https://doi.org/10.1007/s10845-019-01476-x

[338] T. Schlagenhauf and M. Landwehr, "Industrial machine tool component surface defect dataset," *Data in Brief*, vol. 39, p. 107643, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340921009185

[339] J. Fulir, L. Bosnar, H. Hagen, and P. Gospodnetić, "Synthetic data for defect segmentation on complex metal surfaces," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2023, pp. 4423–4433.

[340] W. Matthias, H. Tobias, and H. F. A., "Dagm: Weakly supervised learning for industrial optical inspection [dataset]," 2007. [Online]. Available: https://hci.iwr.uni-heidelberg.de/content/weakly-supervised-learning-industrial-optical-inspection

[341] X. Zhu, T. Bilal, P. Mårtensson, L. Hanson, M. Björkman, and A. Maki, "Towards sim-to-real industrial parts classification with synthetic dataset," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2023, pp. 4454–4463.

[342] Y. Gong, G. Liu, Y. Xue, R. Li, and L. Meng, "A survey on dataset quality in machine learning," *Information and Software Technology*, vol. 162, p. 107268, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584923001222

[343] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, "A brief review of domain adaptation," in *Transactions on Computational Science and Computational Intelligence*, 2021.

[344] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[345] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training GANs," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf

[346] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf

[347] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 586–595.

[348] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, "A review on generative adversarial networks: Algorithms, theory, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3313–3332, 2023.

[349] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *CVPR 2011*, 2011, pp. 1521–1528.

[350] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," 2019.

[351] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Curriculum Vitae

Lovro Bosnar

lovro.bosnar@itwm.fraunhofer.de

Education

2020—2024

Degree: Doctor of Engineering in Computer Science
Where: RPTU Kaiserslautern, Germany
Topic: Procedural Modeling and Image Synthesis for Virtual Surface Inspection Planning
Supervisor: Prof. Dr. Hans Hagen
Focusing on procedural surface modeling for image synthesis for virtual surface inspection planning. During the research conducted for the thesis, I explored the research topics through the development of photo-realistic modeling and image synthesis environment for surface inspection planning. The environment was applied for surface inspection planning and the creation of training-ready synthetic datasets for Machine learning in surface quality inspection.
Highlighted focus areas: Computer graphics, procedural geometry modeling, material modeling, procedural texture modeling, reflection models, path-tracing rendering, physically based models.

2017—2019

Degree: Master of Science in Computer Science
Where: University of Zagreb, Croatia
Faculty of Electrical Engineering and Computing

During master's studies I worked on several projects and attended courses in areas of Computer Graphics, Computer Vision and Machine Learning

- **Highlighted courses:** Computer Graphics. Machine Learning. Computer Vision.
- **Master Thesis:** Modelling Spatial Development of Biological Systems.
  - I have used procedural and generative methods to simulate and animate growth/morphogenesis using Blender and Python.
  - Thesis: `https://urn.nsk.hr/urn:nbn:hr:168:055831`.
  - Code: `https://github.com/lorentzo/Growth_models`
- **Computer Graphic course project:** Designing and developing an artistic model of the Solar System.
  - I have used fractals and procedurally generated textures. Technology: C Sharp and Unity.
  - Code: `https://gitlab.com/lorentzo/computer-graphics-labs`
- **Computer Vision course project:** Optical flow estimation using gradient methods.

- Group project. I have implemented an optical flow algorithm. Technology: Python and OpenCV
- Code: `https://bitbucket.org/eigenvaluer/racunalni_vid/src/master/`

February 2018— July 2018

Degree: Master of Science in Computer Science
Where: University of Science and Technology (AGH), Poland

During my semester abroad (Erasmus+) I have worked on several projects and attended courses in Computer Vision and Machine Learning:

- **Highlighted courses:** Fundamentals of Data Science. Digital Image Processing and Vision Systems.
- **Data Science course project:** Kaggle competition: implementing a model for audio clip classification.
  - I have extracted features using MFCC and applied K-means and SVM. Technology: Python
  - Code: `https://github.com/lorentzo/DS-PROJECT`

2014— 2017

Degree: Bachelor of Science in Computer Science
Where: University of Zagreb, Croatia
Faculty of Electrical Engineering and Computing

During my Bachelor studies I worked on several projects and attended general computer science courses:

- **Highlighted courses:** Programming and Software Engineering. Interactive Computer Graphics. Artificial Intelligence.
- **Bachelor Thesis:** Analysis of Algorithms for Finding Maximum Flow in a Graph.
  - Analysis and comparison of different algorithms for maximum flow in a graph
  - Thesis and code: `https://urn.nsk.hr/urn:nbn:hr:168:431927`

Research

2020— Present

Project: Procedural Modeling and Image Synthesis for Virtual Surface Inspection Planning
Where: Fraunhofer ITWM, Kaiserslautern, Germany
Advisor: Petra Gospodnetic, Markus Rauhut
Contributions:

- Development of photorealistic image synthesis pipeline for virtual surface inspection planning.
- Established a set of general requirements for image synthesis in surface inspection planning
- Development of procedural texturing methods for representing machining surface topologies for industrial quality inspection

- Established a set of general requirements for texture synthesis in surface inspection planning
- Development of procedural geometrical defecting workflow for representing defects occurring in manufacturing for industrial quality inspection
- Established a set of general requirements for geometrical defect modeling in surface inspection planning
- Set foundations for synthetic dataset generation workflow for Machine learning quality inspection

Experience

January 2023.— April 2023.
Position: Computer Graphics Teacher
Where: DHBW Mannheim, Germany

Teaching Foundations of Computer Graphics. My work included:

- Writing syllabus and lectures from scratch,
- Writing projects and exams,
- Giving lectures,
- Correcting projects and exams

June 2020.— September 2020.
Position: Computer Graphics (rendering) Internship
Where: AppleseedHQ, remotely

Google Summer of Code with appleseed. Implementing enhanced normal mapping in appleseed rendering engine based on "Microfacet-based Normal Mapping for Robust Monte Carlo Path Tracing" paper by Vincent Schüßler (KIT), Eric Heitz (Unity Technologies), Johannes Hanika (KIT) and Carsten Dachsbacher (KIT).
Full report and code:

- Report: `https://github.com/lorentzo/GSOC-2020-Report`
- Code: `https://github.com/appleseedhq/appleseed/pull/2886`

August 2019— October 2019
Position: Computer Graphics Internship
Where: Fraunhofer ITWM, Kaiserslautern, Germany

Computer graphics modeling and rendering for surface inspection. My tasks included:

- Building image synthesis pipeline using appleseed rendering engine
- Simulating the inspected object, camera and illumination in a 3D scene

- Synthesizing photo-realistic images using appleseed rendering engine.

October 2018— **Position:** Machine Learning Teaching Assistant
February 2019 **Where:** University of Zagreb, Croatia
Faculty of Electrical Engineering and Computing

Laboratory Assistant for Machine Learning University course. My tasks included:

- Machine Learning methods application in laboratory work
- Helping students, discussing problems and solutions. Examining laboratory assignments.

August 2017— **Position:** Testing Engineer
October 2017 **Where:** Ericsson Nikola Tesla, Zagreb, Croatia

Reviewing low-level telecommunication systems written in C. Testing via TCL. Programming in pairs.

Publications

1. **Lovro Bosnar**, Doria Saric, Siddhartha Dutta, Thomas Weibel, Markus Rauhut, Hans Hagen and Petra Gospodnetic, "**Image synthesis pipeline for surface inspection**", LEVIA 2020: Leipzig Symposium on Visualization in Applications, Leipzig, Germany, *doi:* 10.31219/osf.io/kqt8w
2. **Lovro Bosnar**, Markus Rauhut, Hans Hagen and Petra Gospodnetic, "**Texture synthesis for surface inspection**", LEVIA 2022: Leipzig Symposium on Visualization in Applications, Leipzig, Germany, *doi:* 10.36730/2022.1.levia.4
3. **Lovro Bosnar**, Hans Hagen and Petra Gospodnetic, "**Procedural defect modeling for virtual surface inspection environments**", 2023. IEEE Computer Graphics and Applications, 43(2), 13-22., *doi:* 10.1109/MCG.2023.3243276
4. Juraj Fulir, **Lovro Bosnar**, Hans Hagen and Petra Gospodnetić, "**Synthetic Data for Defect Segmentation on Complex Metal Surfaces**", 2023. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4423-4433)., *doi:* 10.1109/CVPRW59228.2023.00465
5. Juraj Fulir, Natascha Jeziorski, Tobias Herffurth, **Lovro Bosnar**, Katja Schladitz, Henrike Stephani, Thomas Gischkat, Claudia Redenbach, Hans Hagen and Petra Gospodnetic, "**SYNOSIS: Image synthesis pipeline for machine vision in metal surface inspection**", *To be published*
6. Katja Schladitz, Cladia Redenbach, Tin Barisin, Christain Jung, Natascha Jeziorski, **Lovro Bosnar**, Juraj Fulir and Petra Gospodnetic, "**Simulation of microstructures and machine learning**. In Proc. CMDS14, Paris, 2023, Springer Proceedings in Mathematics and Statistics, Eds. A. Cherkaev, D. Jeulin, J. Dirrenberger, S. Forest, F. Willot.

Talks

1. Lovro Bosnar. *Procedural Modeling for Virtual Surface Inspection Planning Environments.* Workshop on "Generating synthetic image data for AI" at the KI 2022 (virtual, hosted in Trier/Germany). Kaiserslautern, Germany, Sept. 2022

Activities and Achievements

- Google Summer of Code Participant 2020
- Student Supervisor - Master students, 2021-2023
- Teacher at DHBW Mannheim: Introduction to computer graphics, 2023
- Mentor - high school student, 2023