Introducing a tool for synthetic defect image data generation: enhancing industrial surface inspection

Ole Schmedemann[a,*], Dirk Holst[a], Jorrid Lund[a], and Thorsten Schüppstuhl[a]

[a]Hamburg University of Technology, TUHH, Institute of Aircraft Production Technology, Denickestr. 17, 21073 Hamburg, Germany

ABSTRACT

Visual surface defect detection in complex scenarios remains a manual endeavor in industrial inspection. State-of-the-art machine learning approaches promise to automate this task through machine vision; however, they require substantial, suitable training data to be effectively trained. For industrial products, such data are often unavailable for several reasons, including the rarity of certain defects or their manifestations. This paper introduces a tool to generate synthetic data for training machine vision systems using rendering techniques. Our tool allows for targeted manipulation of process parameters throughout the rendering chain, including scene composition involving object, camera, and lighting, defect generation, placement, and material generation. This enables an exploration of the impacts along the rendering pipeline and facilitates selecting an appropriate data set for specific inspection tasks. We demonstrate our tool's effectiveness by applying models trained on our synthetic data to a real-world inspection task.

Keywords: Synthetic training data, machine learning, deep learning, visual inspection, industrial quality control, domain randomization, rendering, defect detection

1. INTRODUCTION

Deep learning–based computer vision has recently emerged as a promising approach for automating defect detection in complex industrial inspection tasks. In principle, training deep networks requires substantial quantities of representative image data to capture the range of defect types, appearances, and locations encountered in real-world settings. However, the scarcity of certain defects, especially rare anomalies or those that manifest infrequently, often limits the collection of sufficiently diverse training data sets. This mismatch between the high data demands of deep learning and the limited availability of real-world defect images remains a key bottleneck in deploying machine vision for industrial inspection.

Synthetic image generation via 3D rendering has gained increasing attention to address this challenge. One can systematically generate the training set by creating virtual scenes, placing virtual cameras and lights, and embedding programmatically generated defects. This approach enables engineers to incorporate domain knowledge about plausible defect shapes, sizes, or locations and to exercise complete control over the appearance of surfaces, background, and lighting conditions. Synthetic data can thus complement or substitute real-world images, potentially overcoming data scarcity.

Despite these advantages, the transition from synthetic data to real-world performance is far from trivial. Differences in content and appearance between rendered and real-world images cause models trained purely on synthetic images to perform poorly when applied to real-world applications. Efforts to mitigate this performance gap include realistic rendering pipelines and the complementary domain randomization (DR) strategy, where exaggerated or non-physical variations (e.g., in lighting, textures, or geometry) compel the model to learn domain-invariant features. While feasibility of synthetic data has been demonstrated in numerous defect detection scenarios[1–3], a holistic blueprint how to balance realism and variation for optimal performance for a novel defect detection use case is still lacking.

Further author information:
* E-mail: ole.schmedemann@tuhh.de

For a given defect detection use case, it remains unclear which pipeline parameters should be systematically varied to identify their contributions to the performance of the rendered data set. Therefore, it is necessary to systematically investigate which parameter variations to introduce and to what extent to develop a sufficiently representative data set.

This work first analyzes which specific rendering pipeline parameters may influence the final model's performance in a defect detection task. Next, we propose a controllable data generation pipeline that allows systematically tuning these parameters and to investigate which configurations matter most for a task. These pipeline parameters include composition models of scene objects, defect generation characteristics like size, positioning, and shape of a defect model, material parameters, camera parameters, and lighting parameters. Building on this pipeline, one may explore pipeline parameters systematically to optimize model performance. Finally, we generated a synthetic data set with the pipeline for a real-world use case to demonstrate that the pipeline generates a variation-rich data set. A model trained on this data generalizes effectively to real-world conditions, achieving a mAP@0.5IoU of 0.87 on the real-world validation set.

2. RELATED WORK

In this chapter, an overview of synthetic data generation approaches for visual defect tasks is presented. We refer to synthetic images in general when images are computer-generated or significantly manipulated by computers. Different methods for generating synthetic image data for application in computer vision are used, which can be roughly grouped into *generative learning methods*, *rule-based 2D image composition*, and *rule-based 3D rendering*.

Generative learning methods This family of approaches uses machine learning models, such as Generative Adversarial Networks (GANs), Variational Autoencoders, or diffusion models, to create synthetic images. In the context of industrial inspection, GAN-based schemes are utilized to insert defects into nominal (defect-free) images, effectively augmenting limited data sets.[4–9] Other works employ convolutional autoencoders (CAEs) for a similar purpose.[10,11] Some recent studies explore text-to-image diffusion probabilistic models (DPMs)[12], such as DALL-E[13] or Imagen[14], to generate training images from textual prompts.[15]

Despite the often highly realistic visual quality these methods can achieve, they have several drawbacks when used for defect detection. First, they rely on training data of the same domain, which does not solve the fundamental problem of having few or no real-world defect images. Second, they limit user control over specific defect parameters such as size, depth, or exact location and thus cannot easily incorporate extensive domain knowledge. On the other hand, these models are prone to unwanted artifacts. Third, because the generation mechanism is image-based, the method does not automatically produce pixel-level ground-truth annotations: one must rely on manual or semi-automatic labeling to create large-scale training sets. Consequently, many authors view ML-based generation instead as a form of advanced data augmentation rather than a general synthetic data solution.[16]

Rule-based 2D image composition Image composition approaches manipulate two-dimensional image data to create synthetic variations. For instance, *cut-and-paste*[17] strategies superimpose new objects, background regions, or defect patches onto existing images.[18–20] Other examples are procedurally generated images[21], manipulating real-world images, or combining real-world and procedurally generated images[22,23].

While these rule-based techniques are straightforward to implement, they can struggle to produce physically accurate lighting, shadows, or material changes. A recurring challenge is the inability to simulate geometric deformations. If a defect implies missing or protruding material, simple 2D overlays will not account for occlusions or changes in shading. Moreover, when an object's shape is significantly altered, 2D compositing cannot reveal what lies behind newly created holes or cracks. As a result, rule-based 2D rendering often suffices for minor surface anomalies but fails to capture more complex defects or detailed lighting interactions.

Rule-based 3D rendering Rule-based 3D rendering addresses these shortcomings by constructing a synthetic scene comprising 3D geometry, camera, and lighting, and rendering it. This technique can yield high-fidelity images and automatically produce pixel-accurate annotations. Although it requires more effort, 3D rendering provides greater realism and better handles occlusions, highlights, and shadows.

3D rendered data sets have been produced for various applications, from object detection[24–26] to robotic vision[27]. General purpose toolboxes such as Kubric[28], BlenderProc*[29], NDDS†, CAD2Render[30], Infinigen[31] and Perception‡ ease the process of generating annotated images. These frameworks typically assume that accurate 3D models are readily available. Although CAD models of the objects under investigation are often available for industrial inspection tasks, these models represent only the defect-free state. Consequently, they cannot be used directly to generate images of defective parts.

2.1 Minimizing the Domain Gap

Synthetic image data can differ significantly from real-world images, leading to reduced performance of models trained with this data when applied to real-world data. This observation is commonly referred to as *domain gap*[32], *sim-to-real-gap*, or *reality gap*. This is due to rendering engines only approximating real-world cameras, and real-world light interactions with surfaces, and that 3D models are solely approximations of their real-world counterparts (appearance gap). Furthermore, synthetic objects and scenes are often idealized or have reduced detail compared to real scenes, and the probability of occurrence and placement of objects does not correspond to the real world, sometimes referred to as the content gap. Two main strategies are used to mitigate this issue:

2.1.1 Realistic simulation

One approach is to push simulation fidelity as far as possible by using detailed 3D models, advanced material shaders, complex lighting setups, and even sensor-specific artifacts. Ref. 33 shows that adding more realistic materials can improve object recognition performance. In robotics, physically based simulators for self-driving or manipulation tasks also adopt high-quality renders to reduce the sim-to-real gap.[34]

However, realism alone can be costly and may risk overfitting to the specific synthetic environment. Minor errors in reflectance models or missing micro-geometry might lead to biases that neural networks exploit in synthetic data but fail on real-world test images.

2.1.2 Domain randomization

Domain randomization (DR) is a technique that generates synthetic image data with a wide array of randomized visual properties to serve as training material for deep learning-based computer vision models. By systematically varying elements such as textures, lighting conditions, object placements, and background details, this approach compels models to focus on underlying, intrinsic features rather than superficial visual cues. In doing so, the model learns to extract a domain-invariant representation that remains robust and effective even when confronted with the variability and unpredictability of real-world data.[24,35,36] An advantage of this approach is scalability: because the focus is on coverage rather than perfect realism, generating large quantities of randomized images is often faster and less resource-intensive than meticulously crafting photorealistic scenes. When randomness is structured (i.e., physically plausible, context-aware), models can still reap the benefits of coverage while preserving realism.[37]

Effective DR pipelines often blend physically accurate rendering (materials, shadows) with broad variation (texture randomization, camera jitter, lighting changes). The consensus is that the ideal balance between sensor realism and randomization is highly application-specific, requiring empirical tuning.[34]

2.2 Synthetic Data for Defect Detection

A growing body of research focuses on specialized pipelines for defect image synthesis. One can differ between the underlying defect introducing principle used: While geometry-based methods directly alter the 3D mesh, texture-based methods[3,38–48] modify surface appearance without significantly changing the mesh geometry.

*https://github.com/DLR-RM/BlenderProc

†https://github.com/NVIDIA/Dataset_Synthesizer

‡https://github.com/Unity-Technologies/com.unity.perception

2.2.1 Geometry-based defect generation

An example of a geometry-based defect generation is Ref. 49. They introduced a procedural crack model that subtracts geometry along fracture lines. Early work by Ref. 50,51 used parametric shapes and physics simulations to produce realistic glass shards, with cracks adapted from prior procedural models.[52] Similarly, Ref. 53 advocates a structured pipeline of 3D simulation and parametric modeling to recreate cracks on microchips.

Ref. 1, 54, 55 propose a pipeline that programmatically creates flaws in parts by Boolean operations in the object's mesh. These geometry modifications (e.g., dents, scratches) are rendered under physically based illumination, yielding photo-realistic images. In previous work, we introduced a similar approach to imprinting defect negative objects in cast-iron parts and extensively randomizing lighting and camera parameters extensively.[56] Recent research by Ref. 57, 58 explores multi-view setups, emphasizing the importance of capturing consistent defect geometry across different viewpoints. Meanwhile, Ref. 2 systematically varies mesh changes to represent flash defects on injection molded parts, highlighting that camera and lighting variation can profoundly affect the final model performance even for a single defect type.

The main advantage of geometry-based approaches is physical plausibility: lighting, self-occlusion, and shadows emerge naturally from the shape modifications. However, modeling complex material removal or cracks may be computationally demanding.

2.2.2 Texture-based defect generation

Texture-based approaches either modify the material, thereby locally altering its reflectance properties and creating deviations from the defect-free regions, or they adjust the geometry exclusively during rendering, for example by employing textures for virtual displacement mapping[59]. Previous work leverages 2D textures or shader manipulations to represent defects like cracks, stains, or corrosion. For example, Ref. 38 generates procedural defect textures (cracks, foreign inclusions) on a steel surface, applying local displacement maps to mimic slight geometric disturbance. Ref. 39 overlay defect masks on a car door's texture, randomly varying roughness and normal maps. Similarly, Ref. 40 combines photogrammetry with photometric stereo to capture real surface textures, then adds blowhole-like defects via local noise-based displacement.

Texture-based methods often suffice for small or shallow defects, such as scratches or corrosion, and can be more efficient than large Boolean operations. However, they cannot accurately depict scenarios where significant geometry changes reveal occluded areas (holes, breakages) unless combined with partial geometry editing. Several authors thus advocate a hybrid pipeline, where texture-based shading changes handle subtle scratches, and mesh-based operations handle bulk deformations.[3, 46]

2.2.3 Open challenges

Despite these advancements, open questions remain. First, many works use synthetic data to show anecdotal or case-specific success but do not systematically quantify how each pipeline parameter (e.g., lighting, mesh detail) contributes to final accuracy. It was not examined holistically how to consider and analyze all potentially influencing steps in generating a synthetic data set. Pipelines are often tailored to a single part geometry or defect type. Generalizing to new surfaces, new lighting configurations, or new defect physics can require non-trivial engineering.

2.3 Summary and Our Contribution

Efforts to minimize the domain gap center on *realistic simulation* and *domain randomization*. However, an optimal combination is often application-specific and best found via empirical experimentation. For industrial surface inspection, both geometry-based and texture-based defect creation strategies have demonstrated feasibility, though most works address a narrow set of defect types or geometric settings.

In this work, we aim to analyze all relevant factors influencing the performance of models trained on synthetic image data sets generated via 3D rendering for defect detection. We provide a holistic overview of the data generation pipeline, from the rendering scene setup to subsequent post-processing steps. We highlight how each step contributes to the final data set quality and model performance. *Our proposed implementation* draws on the experience of these prior studies by combining physically based rendering with a modular structure that lets

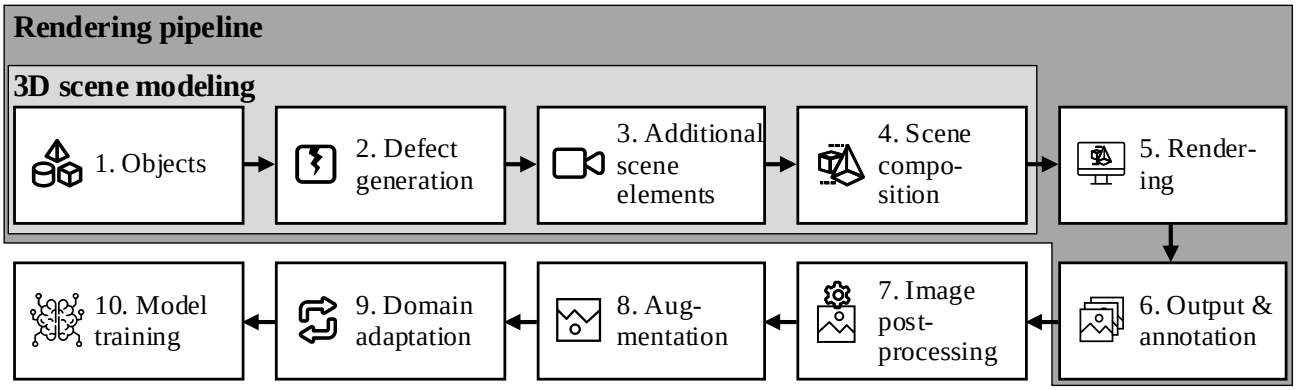


Figure 1. Schematic representation of the synthetic data set generation workflow, incorporating image synthesis steps that ultimately influence the final model's performance in defect detection.

the user tune or randomize factors like geometry, material properties, lighting, camera parameters, and defect properties. This structured pipeline is intended so that a user can systematically vary the realism or degree of variation of synthetic data sets, and it allows for comprehensive ablation experiments. In this work, we focus on geometry-based defect generation. However, the proposed pipeline can be extended to include texture-based defect generation.

3. ANALYSIS OF INFLUENCING PROCESS STEPS

The degree of realism and domain randomization required varies by application and must be determined case-by-case. To generate a suitable synthetic data set, researchers and practitioners must understand which image characteristics can potentially enhance model performance and employ a tool capable of producing data sets with these features. Building on the clustering approach described by Paulin et al.[16], we adapt the categorization of all potentially influencing steps along the image generation pipeline specifically to defect generation, see Fig. 1. Each step influences the appearance of the images and thus may affect the quality and suitability of the final data set.

For the rendering pipeline, these steps include origin and manipulation of scene objects, generating a defect, additional scene elements like cameras and lighting, composing all objects to a 3D scene, setting shader properties for all objects, and finally rendering an image. Additional post-rendering steps include 2D image post-processing, which is sometimes used to mimic imaging errors like vignetting or distortions, image augmentations, which include, among others, simple geometric transformations to introduce more variation, and domain adaptation, where image-to-image machine learning models are used to increase realism of renders. In this chapter, we present the influencing properties of the steps of the rendering pipeline (dark grey).

Objects In industrial production, CAD models of objects or entire environments are often readily available; however, when they are missing due to incomplete product variants, lack of OEM-provided designs, or the absence of detailed semi-finished product data, 3D scanning or photogrammetry can fill the gap by reconstructing an object's geometry from multiple photographs or through structured-light and laser-based measurements. Though both methods can produce accurate models, they typically require controlled conditions, specialized equipment, and post-processing. Photogrammetry may struggle with reflective or transparent surfaces, while structured-light and laser scanners handle them better. When background details have minimal impact on defect appearance, omitting detailed 3D models of additional objects and using random background environment maps (e.g., HDRI images) can be a cost-effective way to vary the scene and reduce false positives outside the primary object of interest.

Besides the presence of objects and the choice of background environment maps, the level of detail of objects directly influences their appearance in rendered images. If details present on the real-world parts are not present

on the virtual object, the model may not learn to distinguish between them and the defect and may recognize them as false positives.

Defect Generation Geometry-based methods directly modify the geometry of an object, resulting in more realistic lighting and shadows when the viewpoint or illumination changes. For this approach, a defect model is either subtracted or added to the main object's geometry using Boolean operations to simulate the defect's geometric appearance. Typical examples include dents, nicks, cracks, missing geometry, or any other defect that significantly alters the shape of an object. Object-based defect generation steps consist of the defect object's generation, positioning, manipulation, and imprinting. Defect objects may originate from manual modeling, procedural modeling, physical modeling, or 3D-scanning of real-world samples. Both the generation steps and the origin of objects significantly influence the appearance of the defect. Thus, it must be possible to examine their impact.

Additional Scene Elements Modern rendering engines typically employ a pinhole camera model and advanced lighting simulation to replicate real-world optical conditions. These components can be configured and randomized to enhance the fidelity and variability of synthetic data sets. Assumingly, for virtual cameras, the most influential properties are image resolution and focal length. These parameters directly influence the size of scene objects in terms of pixels.

Accurate lighting involves identifying real-world light sources and measuring or modeling their intensity, color temperature, and light spread angle. Since light models only approximate the behavior of their real-world counterparts, systematically introducing variation to generate slightly different lighting scenarios with each image generation iteration is expected to be beneficial. Therefore, the rendering pipeline should allow for systematically examining different intensities, color temperatures, and light spread angles.

Scene Composition Scene composition refers to the arrangement of cameras, lighting sources, and objects within a virtual environment. It can be defined manually by replicating a known real-world setup, generated procedurally using rule-based algorithms that systematically introduce variations, or created using a combination of both approaches. Procedural methods are particularly advantageous for producing large, diverse data sets, as they automate realistic fluctuations in object placement and camera and lighting viewpoints. Scene composition highly influences a model's performance because it determines which objects are visible in a render, at what size, to what extent illuminated, and from which perspective they are viewed. Consequently, variation in scene composition can be utilized to introduce various defect appearances within a data set.

Rendering Based on the composed scene, rendering is the final step in the rendering pipeline, producing a 2D image from a virtual 3D scene. Modern rendering algorithms commonly solve the *rendering equation*, which describes light transport through a scene. Material properties are represented using a Bidirectional Scattering Distribution Function (BSDF), while ray-tracing or path-tracing methods simulate how light rays interact with surfaces. A simplified pinhole camera model then captures these interactions from a specified viewpoint.

Shading computes a surface's color or intensity based on lighting, material properties, and viewing direction. Modern pipelines often use physically based rendering (PBR), which employs accurate BSDFs, microfacet distributions, Fresnel effects, and energy conservation to achieve realistic reflections and scattering. Rendering engines like Blender Cycles, Unreal Engine, or Pixar RenderMan can efficiently produce high-resolution PBR-shaded images on consumer hardware, making PBR the favored choice for realistic synthetic data generation.

In physically based rendering (PBR), a material specifies realistic surface behavior through a few key parameters: base color (albedo), metallicity, roughness, and normal maps. These parameters ensure that surfaces exhibit believable reflectance and scattering under varying lighting and viewpoints. PBR materials remain consistent across various lighting conditions and viewing angles.

Textures refine the material's appearance by mapping values (e.g., color or roughness) across an object's surface. They may be traditional image-based textures applied using UV coordinates, prone to seams or visible tiling, or procedural textures generated algorithmically and can cover surfaces seamlessly.

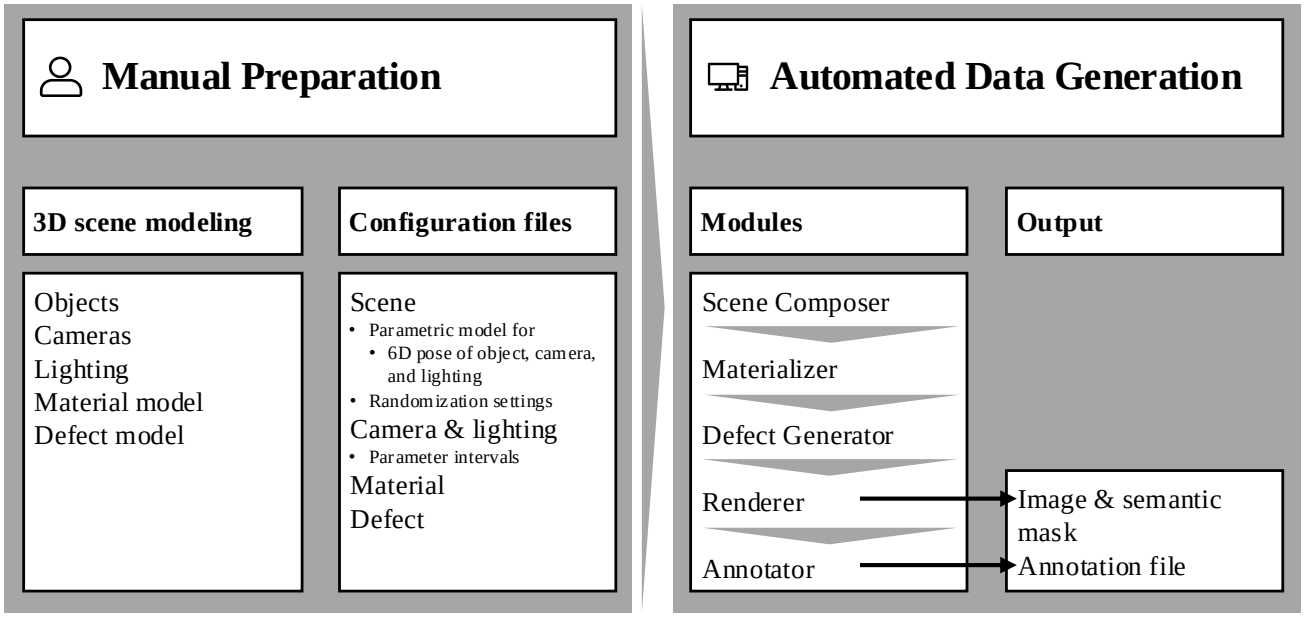


Figure 2. Schematic overview illustrating the training data generation process.

To what extent realistic reflection behavior contributes to a final model's performance remains an open question. Procedural approaches offer systematic parameter variation, which may be valuable for creating highly diverse defect appearances in a synthetic data set. Therefore, a rendering pipeline should incorporate selecting different object materials or adjusting material properties within given boundaries to systematically introduce variation in appearance and address the appearance gap by forcing the model to learn material-invariant defect features.

Output and Annotation A central advantage of synthetic image generation is producing pixel-accurate annotations for training and evaluating machine learning models. In object detection tasks, for example, standard annotation formats are YOLO, COCO[60], or Pascal VOC[61]. The essential requirement is to assign each pixel in the render to either the background or one of the defect classes, so that bounding boxes or segmentation labels can be extracted automatically. Annotations must mark what is to be detected in the image, for example, if something is broken off due to a defect, the aim can either be to detect the break-off edge or the missing geometry. Depending on the task, the tool must provide a suitable annotation method.

4. AUTOMATED RENDERING PIPELINE FOR DEFECT IMAGE DATA GENERATION

Building on the analysis introduced in Sec. 3, we have implemented a modular, automated rendering pipeline in Blender[§] that systematically generates annotated defect image data. This pipeline encompasses geometry-based defect generation methods and supports parametric scene composition, material randomization, and automated annotation. In contrast to our prior work introduced in Ref. 56, the current pipeline incorporates additional modules for procedural scene composition and parametrized material randomization.

Fig. 2 provides an overview of the overall workflow. In a preparatory step, the user models a *task-specific 3D scene*, including all relevant objects (e.g., the inspected part and background items), material models, virtual cameras, and illumination sources. Once this setup is complete, the pipeline automatically generates a series of unique scenes and renders corresponding images, along with pixel-accurate ground-truth annotations indicating a

[§] https://www.blender.org/

defect's location. Each module is controlled through user-defined configuration files, and a guiding script orchestrates their execution in sequence. Repeating this sequence yields the desired data set size. The configuration files define how each module in the pipeline behaves, specifying parameters such as camera placement strategies, material properties, and defect characteristics. To systematically investigate how substeps affect the model's performance, the user modifies the configuration files. When working with a fixed random seed, the rest of the pipeline repeats its behaviour, only reflecting the changes in the configuration.

4.1 Scene Composer

The *Scene Composer* module begins with a manually prepared 3D scene. In this scene, users place or model all relevant objects (including the object under inspection, all pertinent background geometry, cameras, and lights) at approximate or canonical locations. This step ensures that the digital environment broadly reflects the real-world setup, including camera resolution, focal length, and lighting configuration.

In the scene configuration file, a parametric model for all scene objects, cameras, and light sources, whose placement in the scene shall be varied, may be chosen to define a set of allowable poses. The Scene Composer module samples a translation pose from the model in each iteration to mimic sufficient coverage of real-world pose variation.

Parametric models implemented are a spherical shell model, where allowable positions are defined by the min and max limits for the spherical coordinates radius, elevation angle, and azimuth angle, and a cylindrical shell model with min and max limits for cylindrical coordinates. The module can be easily extended to support additional parametric models. Orientation of the objects can either be centered to a fixed pose or again sampled from a set of poses in the world coordinate system. Alternatively or additionally, for each object, constraints for allowable rotation around its local axes can be set for the module to sample from.

After the Scene Composer applies all specified parametric translation and orientation to the scene objects, further scene objects' parameters may be sampled within user-specified intervals. By default, this includes lighting intensities, lighting colors, camera resolutions, and camera focal lengths that can be randomly sampled within physically plausible ranges. For background variation, environment maps can be sampled with each iteration for a given set of environment maps.

The plausibility of the scene composer configuration should be visually validated in advance. With the tool, users can test and visualize positional and parameter sampling before data set generation to tune a reasonable configuration.

4.2 Materializer

In the preparation phase, a material model must be defined or developed for each object in the scene. Our Materializer module utilizes user-defined procedural material models, which must incorporate a BSDF model and detailed geometric topology through displacement mapping. To enable variation in the material's appearance, the model must be parametrized at a high level so that users can specify parameter intervals for the module to sample from during each image generation iteration. Examples of these high-level parameters include scaling, translating, and rotating the material mapping on an object and adjusting input parameters for noise functions (e.g., Perlin noise) used to modulate color, roughness, and other material properties.

Once the scene layout is finalized, the Materializer module assigns and, if desired, samples parameters for the material models of each object in the scene. This approach facilitates systematic variation in surface finishes, reflectivities, and colors across images.

4.3 Defect Generator

For geometry-based defects, the Defect Generator begins by selecting an appropriate location on the surface of the inspected part. The user specifies a set of candidate areas, each representing a plausible defect location. Algorithm 1 details how a point is uniformly chosen from these candidate areas while ensuring it remains within the camera's field of view and is not occluded by other geometry. Similar to the orientation constraints in the Scene Composer module, the defect object's orientation relative to the surface normal at the selected location

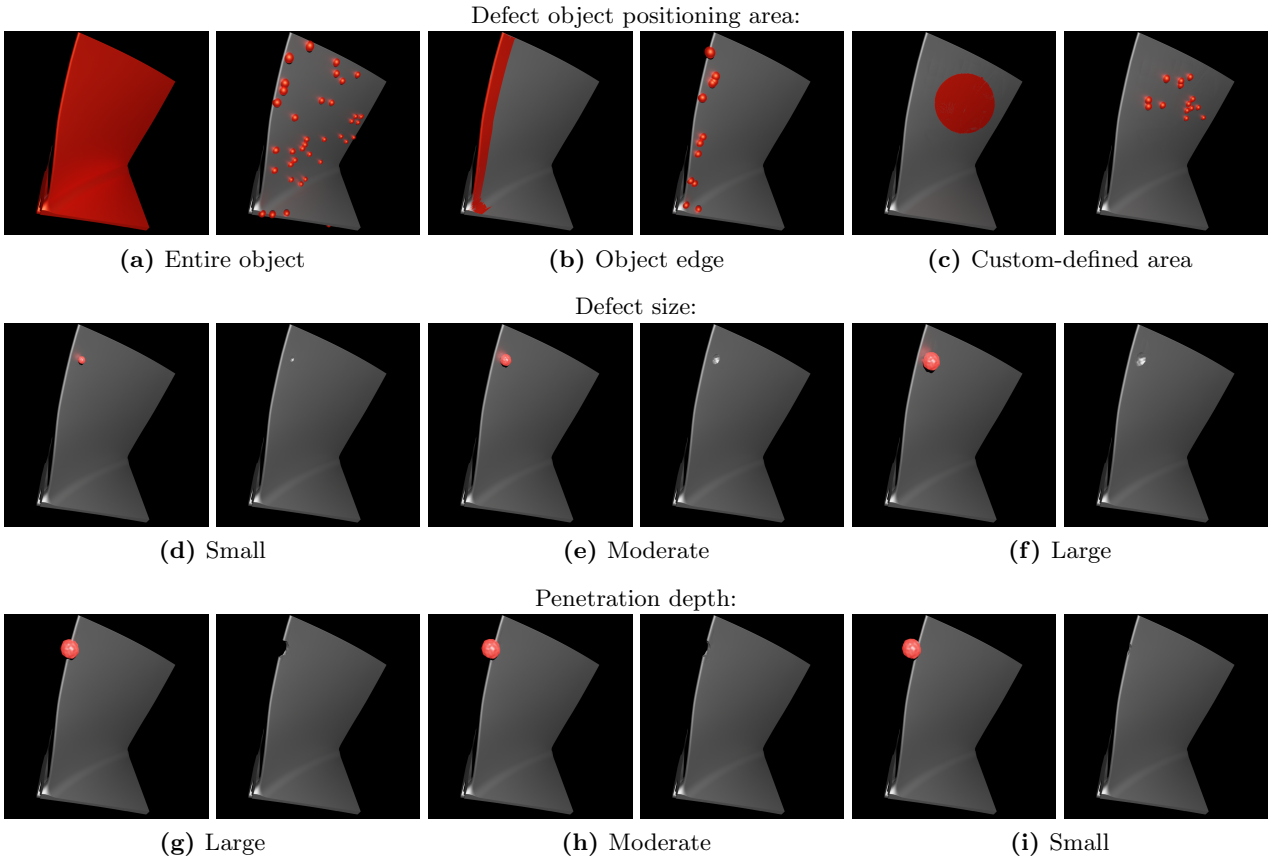


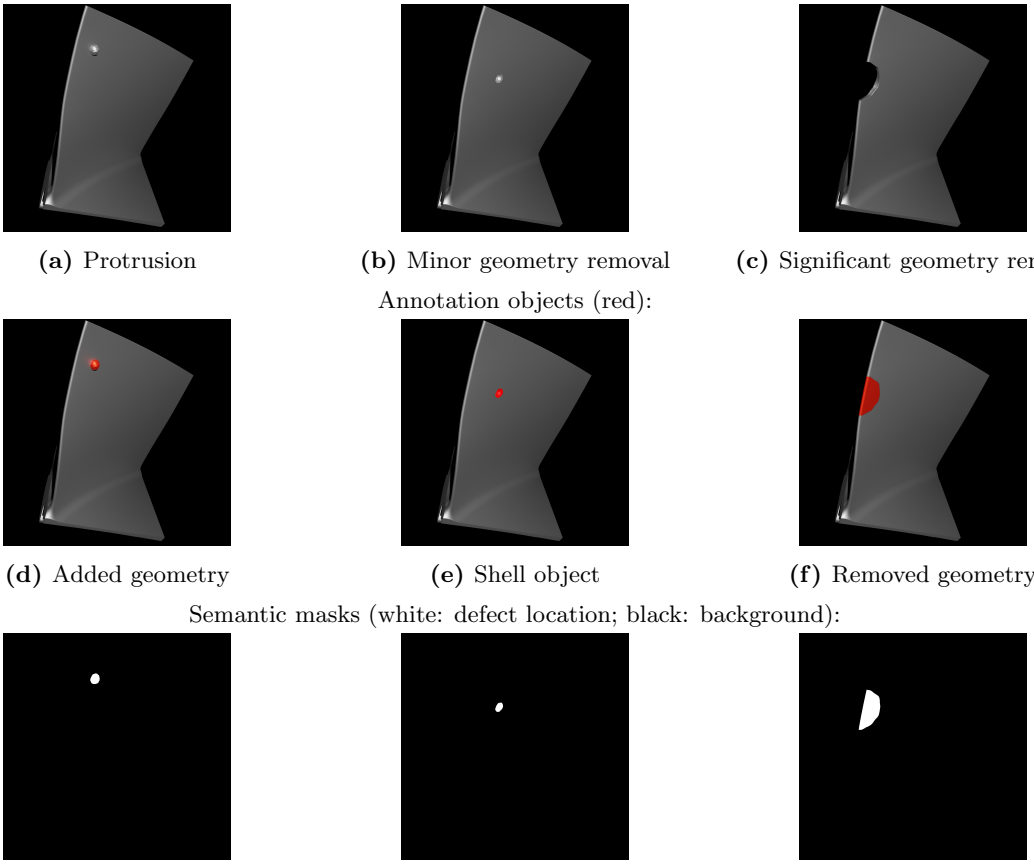
Figure 3. Example renderings of a turbine blade designated for inspection, illustrating user-defined defect object positioning areas (top row), the effects of varying defect object sizes, and different penetration depths for a constant-size defect object displaced along the surface normal. Defect objects are shown as red spheres, while candidate surface areas are highlighted in red.

may be restricted by enforcing user-defined rotation limits along the normal axis. Figures 3a-c illustrate defect positions on an object as red spheres, corresponding to the predefined candidate areas.

After selecting the location, the pipeline places a *defect object* at that point. During the preparation phase, the user must model the defect object, which can be manually created, procedurally generated, or derived from physical simulations or real-world measurements. Figures 4b-c provide examples of such defect objects. To introduce variation along the break-off edge, the defect object may be subjected to user-defined displacement mapping that adds fine details.

Next, the module samples scaling factors and translation distances along the surface normal from user-defined randomization intervals (see Figures 3d-i). Then, a Boolean operation merges the defect object with the main part mesh, simulating a protrusion, a shallow cavity, or significant geometry removal (see Figures 4a-c). Following the Boolean operation, the module automatically creates an annotation object corresponding to the defect. For protrusions, this annotation object is the added geometry itself (see Figure 4d). For shallow cavities or moderate geometry removal, a thin-walled shell object is embedded to delineate the defect region (see Figure 4e). For significant geometry removal, the removed portion of the mesh is preserved as an annotation object (see Figure 4f). In every case, the annotation object remains hidden in the final image but is used to generate an accurate, pixel-wise semantic mask indicating the defect's location.

In summary, the module enables systematic investigation of how defect object shape, size, penetration depth, position, orientation, and topology changes affect the outcome.



(a) Protrusion **(b)** Minor geometry removal **(c)** Significant geometry removal

Annotation objects (red):

(d) Added geometry **(e)** Shell object **(f)** Removed geometry

Semantic masks (white: defect location; black: background):

Figure 4. Demonstration of different automatic annotation strategies. (Left) Added geometry is treated as a separate object. (Middle) A thin-walled shell object is positioned within the cavity following minor geometry removal. (Right) Removed geometry is employed as a separate object but is only rendered to generate the semantic mask (bottom row).

Algorithm 1 Uniform Point Selection on Surfaces

Require: S: set of surfaces
Require: δ: density criterion
Ensure: P: list of selected points
$P \leftarrow \emptyset$ ▷ Initialize an empty list of points
while P does not satisfy the point density criterion δ **do**
    $F \leftarrow$ select a surface from S (based on weighted probability)
    $p \leftarrow$ sample a point uniformly on F
    $P \leftarrow P \cup \{p\}$
end while
for all $p \in P$ **do**
    Check the surface normal orientation at p
    Check if p lies within the camera's field of view
    Check for occlusion of p via raycasting
end for
Select one final point $p_{\text{final}} \in P$
return P ▷ (Optionally, also return p_{final})

4.4 Renderer and Annotator

Finally, the *Renderer and Annotator* modules render the parametric scene with Blender's *Cycles* engine. During this process, based on the annotation object, a semantic mask is rendered (Fig. 4 bottom row). This mask is saved, encoding each pixel whether it shows the background or a defect class. Optionally, the pipeline converts the semantic mask into a user-specified annotation format, such as COCO or YOLO.

5. CASE STUDY AIRCRAFT ENGINE INSPECTION

In this section, the data generation pipeline is implemented demonstratively for a use case from the inspection of aircraft engines. Aircraft engines are routinely subjected to visual inspections to enable early detection of faults. Borescopes inspect internal machine components such as blades and vanes without requiring complete engine disassembly. The borescope is manually inserted into dedicated borescope ports. Once it is in place, the engine shaft is rotated so that the blades sweep past the borescope's field of view, permitting a thorough inspection of each engine stage. Human inspectors then visually detect and classify any defects.

However, automated defect detection and classification present significant challenges. Because the borescope is manually guided, the viewpoint changes slightly with each inspection, resulting in varying perspectives. Furthermore, the borescope is positioned close to the object, so slight rotations or shifts can cause substantial changes in the captured images. Illumination provided by the borescope itself often leads to glare and insufficient lighting. In addition, the miniaturized sensor and optics can introduce limited resolution and imaging artifacts. Shadows cast by objects in the scene further complicate image interpretation, and material variations add another layer of complexity (see Fig. 5).

These challenges have historically rendered conventional image processing unsuitable for this application. Although deep learning–based image processing is well suited to cope with the extensive variations seen during

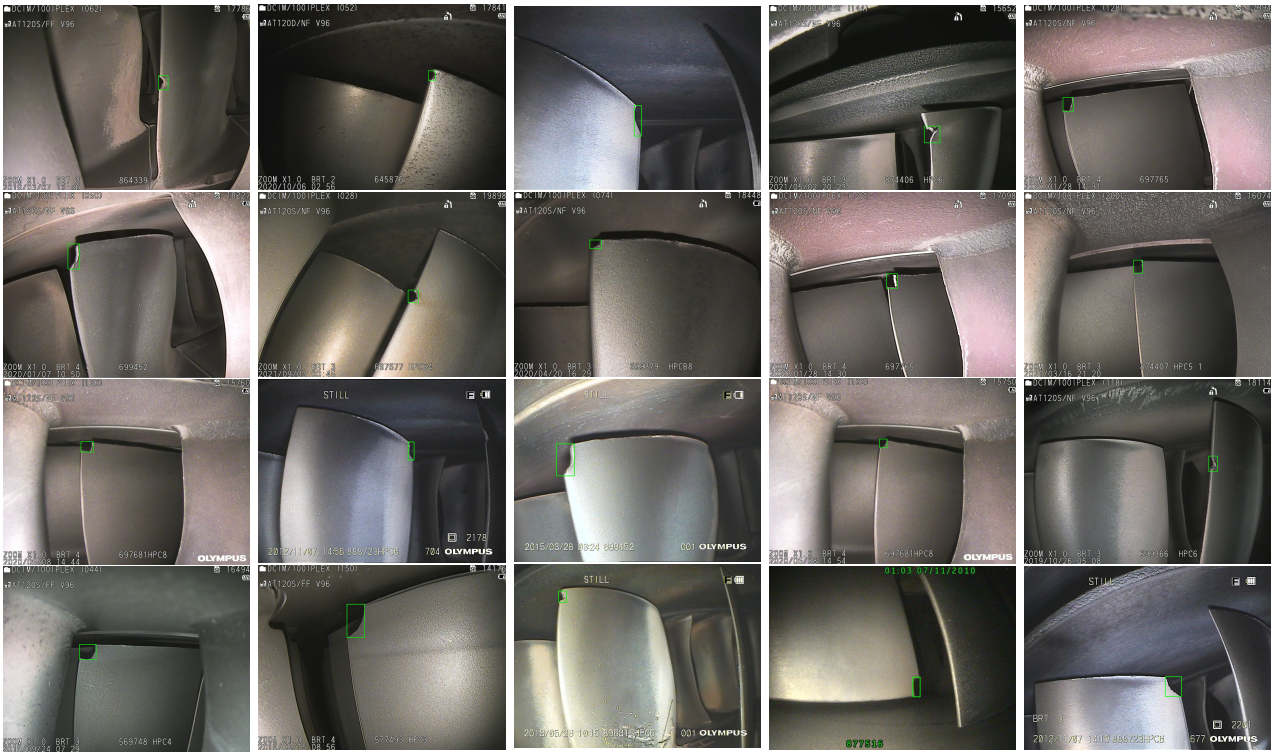


Figure 5. Samples from the real-world data set[62] of borescope inspection of aircraft engines. The illustrated defect category, *missing material*, depicts a blade section that has been removed. Green boxes indicate the ground truth bounding box annotations.

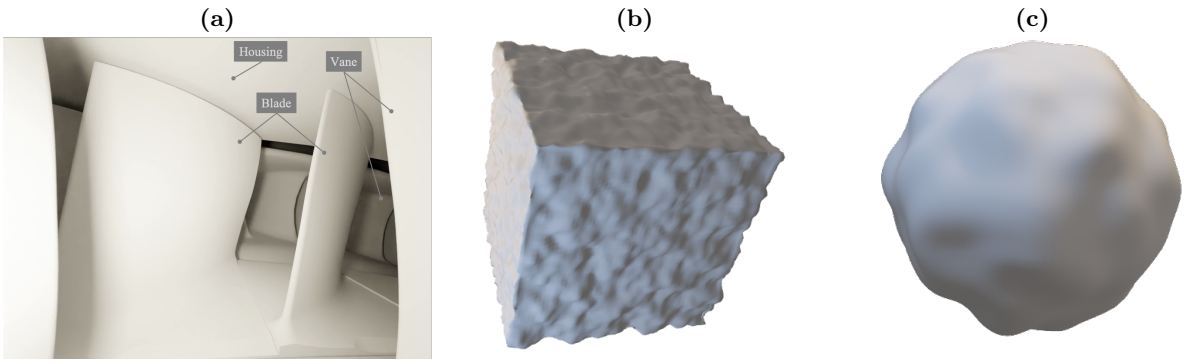


Figure 6. (a) Virtual scene as viewed through a borescope port. (b) Cube-shaped defect object. (c) Sphere-shaped defect object.

borescope inspections, there is a lack of sufficient training data for certain defect classes and their manifestations.[63] This shortage hampers the development of comprehensive and robust models.

One relevant defect class is *missing material*, where a blade has fractured into two or more large segments. For this defect class, Ref. 62 provided a data set of 111 images that were pre-annotated by domain experts (see Fig. 5) and show the missing material defect in the *high pressure compressor (HPC)* section of an engine. We partially refined the bounding boxes to reflect the actual defect manifestations more accurately and used the data set as validation data.

5.1 Synthetic Data Generation

Based on our synthetic data generation procedure introduced in Sec. 4, the 3D scene is modeled in the first step. CAD models of a CFM56 engine, including shafts, blades, vanes, and the housing, were available. These elements were arranged such that all objects visible from the borescope ports of the (HPC) could be observed, as illustrated in Fig. 6a.

For scene composition, the only movable component, apart from the camera and lighting, is the engine shaft, which we allowed to rotate around its central axis. This rotation emulates the actual inspection procedure, where blades on the shaft pass through the borescope's field of view. For the camera's translational parameters, we adopted cylindrical coordinate sampling, as it best captures the real-world movement of the borescope inside the port. To constrain the camera orientation, we specified fixed intervals for rotation around three axes: rotation along the borescope axis and roll and pitch along the camera axis. The lighting's 6D pose was fixed relative to the camera, reflecting the configuration of real borescopes. An area light model with constant intensity and color was used.

We then created a procedural metallic material (see Fig. 7) that allowed parameterization of various base colors and a localized *dirt* color and shape, simulating real-world samples with corrosion or debris. The material's roughness and geometric topology were governed by Perlin noise textures, with input parameters that produce a wide range of appearances. We defined intervals in the material configuration file for all parameters, which the Materializer module used to generate unique material instances for each iteration.

Next, we introduced the defect. Because this defect class involves significant material removal, we represented it as a geometry-based defect, reusing the removed material as an annotation object (see Sec. 4). We assume that the shape of the break-off edge can be approximated by two geometric primitives: a sphere and a cube (Fig. 6c and b). Since the defect predominantly occurs along the leading edge of a blade, that surface region was designated for defect positioning. Finally, we specified intervals for the defect object size and penetration depth along the normal of the surface.

5.2 Model Training and Results

We rendered 500 images (Fig. 7) using the procedure described in the previous section. To evaluate the suitability of this synthetic data set, we trained a Faster R-CNN[64] (pretrained on COCO) and applied augmentations such

Figure 7. Representative samples from the synthetically generated data set using the developed rendering pipeline. These examples illustrate variations in viewpoint, defect position and shape, material appearance, and image resolution. The automatically generated bounding box annotations are shown in green.

as D4 transformations, rotations, bounding box–safe random cropping, downscaling, defocus, blur, illumination changes, salt-and-pepper noise, brightness/contrast adjustments, and hue/saturation shifts. With an SGD optimizer and a batch size of 8, we conducted a 20-iteration Bayesian hyperparameter search for learning rates between 0.0001 and 0.01. The best-performing model achieved a $mAP@0.5IoU$ of 0.87 on the validation set. Fig. 8 shows example results of inference that indicate that the model can transfer to a real-world inspection task.

6. DISCUSSION

Comparing Figs. 7 and 8 qualitatively shows that the proposed pipeline can reproduce variations in real-world image features, including changes in camera viewpoint and perspective, object positioning, and defect shape, size, and position. Quantitatively, a model trained exclusively on the synthesized images achieves performance on par with models trained on real-world samples: for the same data set, Ref. 62 reports an mAP@0.5IoU of 0.93, whereas our approach attains 0.87, indicating that synthetic defect images can effectively substitute real-world data.

The specific choice of pipeline parameters in this case study (e.g., the range of random rotations or permitted defect sizes) was guided by engineering intuition. For future work, the controllable nature of the pipeline enables systematic parameter exploration, allowing more profound insights into which rendering choices most strongly affect model performance. An ablation study, for example, might alter only the camera orientation (while fixing translation) or adjust the lighting angle in small increments. In addition, it remains important to investigate which training strategies confer the most significant advantage, such as mixing synthetic and real-world data sets or fine-tuning models initially trained on synthetic data with real-world data.

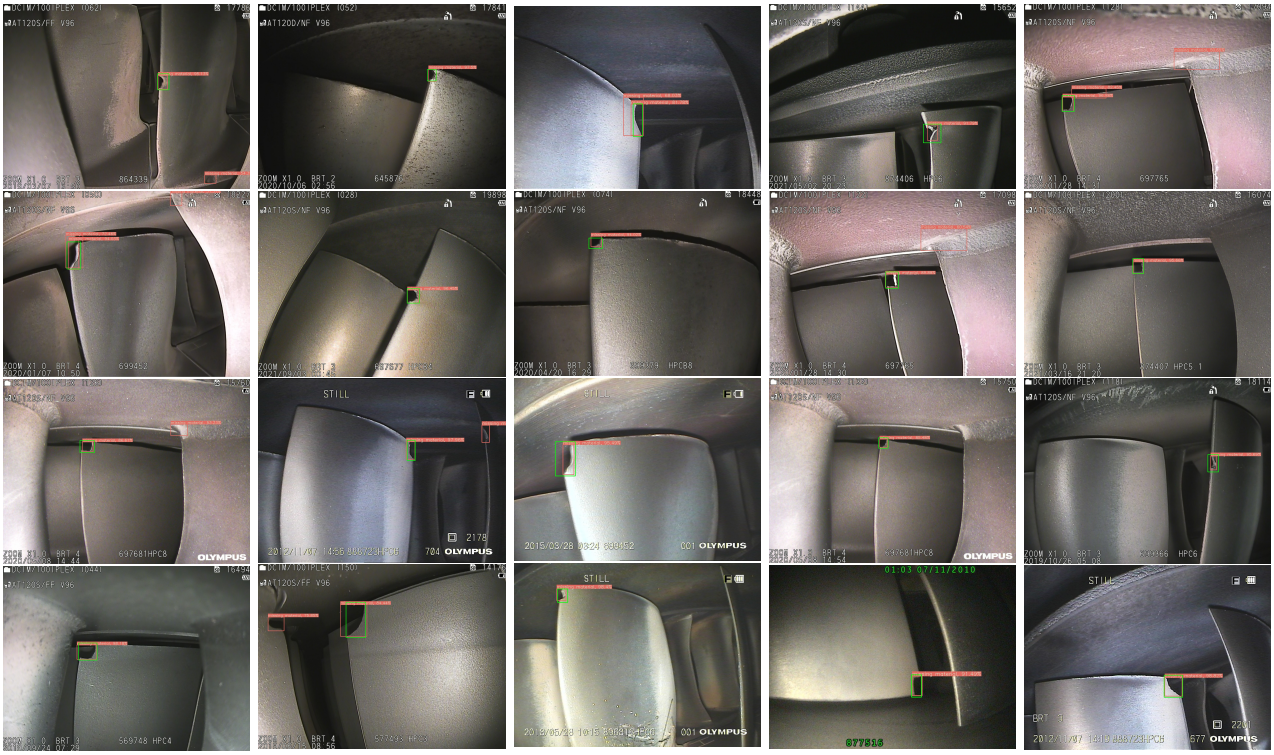


Figure 8. Representative inference outcomes from the model trained exclusively on synthetic data. Ground truth bounding boxes are green, while red boxes indicate the model's predictions.

7. CONCLUSION AND OUTLOOK

We have presented a modular, automated pipeline for generating synthetic images of industrial surface defects and demonstrated its effectiveness on a challenging aircraft engine inspection task. Our approach enables fine-grained control over geometry-based defect creation, material properties, scene composition, and lighting configurations. The pipeline can thus systematically explore each stage of the rendering process to identify the settings that matter most for training defect detection models.

Using a data set of synthetic borescope images, we trained an object detection model to detect *missing material* defects in real-world aircraft engine inspection images, achieving an mAP@0.5IoU of 0.87 without any real-world training samples. These results show that carefully configured synthetic data alone can suffice to train models for industrial defect detection.

In future work, we plan to deploy the pipeline to conduct comprehensive ablation studies to reveal which randomization parameters are most critical for boosting model performance for the presented use case. Additionally, we plan to refine material realism by combining physically based rendering with inverse procedural modeling techniques (e.g., Ref. 65) that derive procedural textures from measured real-world data. Although these methods still require semi-manual inputs and remain susceptible to errors, they offer a promising route to replicating surface appearance more accurately while preserving the ability to introduce tailored variations.

Furthermore, our pipeline's modular architecture extends defect modeling to hybrid texture-based and geometry-based approaches, enabling the efficient representation of minor surface anomalies and large-scale material removal. A key challenge ahead is systematically validating the impact of different pipeline parameters across diverse inspection tasks. Progress in this direction would help establish clear, *a priori* guidelines for synthetic data generation—significantly reducing the engineering overhead and further broadening the industrial adoption of synthetic training data for inspection tasks.

ACKNOWLEDGMENTS

This research was funded by the German Federal Ministry for Economic Affairs and Climate Action under grant number 20Q2109D.

REFERENCES

[1] Bosnar, L., Hagen, H., and Gospodnetic, P., "Procedural Defect Modeling for Virtual Surface Inspection Environments," *IEEE Computer Graphics and Applications* **43**, 13–22 (Mar. 2023). Conference Name: IEEE Computer Graphics and Applications.

[2] Schraml, D. and Notni, G., "Synthetic Training Data in AI-Driven Quality Inspection: The Significance of Camera, Lighting, and Noise Parameters," *Sensors* **24**, 649 (Jan. 2024).

[3] Seiler, F., Eichinger, V., and Effenberger, I., "Synthetic Data Generation for AI-based Machine Vision Applications," *Electronic Imaging* **36**, 276–1–276–5 (Jan. 2024).

[4] Niu, S., Li, B., Wang, X., and Lin, H., "Defect Image Sample Generation With GAN for Improving Defect Recognition," *IEEE Transactions on Automation Science and Engineering* , 1–12 (2020).

[5] Wu, X., Qiu, L., Gu, X., and Long, Z., "Deep Learning-Based Generic Automatic Surface Defect Inspection (ASDI) With Pixelwise Segmentation," *IEEE Transactions on Instrumentation and Measurement* **70**, 1–10 (2021).

[6] Jain, S., Seth, G., Paruthi, A., Soni, U., and Kumar, G., "Synthetic data augmentation for surface defect detection and classification using deep learning," *Journal of Intelligent Manufacturing* **33**, 1007–1020 (Apr. 2022).

[7] Meister, S., Möller, N., Stüve, J., and Groves, R. M., "Synthetic image data augmentation for fibre layup inspection processes: Techniques to enhance the data set," *Journal of Intelligent Manufacturing* **32**, 1767–1789 (Aug. 2021).

[8] Wang, R., Hoppe, S., Monari, E., and Huber, M. F., "Defect Transfer GAN: Diverse Defect Synthesis for Data Augmentation," (Feb. 2023). arXiv:2302.08366 [cs].

[9] De Mitri, O., Frommknecht, A., Huber, M. F., Müller-Graf, F., and Distante, C., "Synthetic data generation for improvement of machine learning-based optical quality control: a practical approach in the welding context," in [*Multimodal Sensing and Artificial Intelligence: Technologies and Applications III*], Stella, E., Ceglarek, D., Kemao, Q., and Soldovieri, F., eds., 46, SPIE, Munich, Germany (Aug. 2023).

[10] Han, Y.-J. and Yu, H.-J., "Fabric Defect Detection System Using Stacked Convolutional Denoising Auto-Encoders Trained with Synthetic Defect Data," *Applied Sciences* **10**, 2511 (Apr. 2020).

[11] Yun, J. P., Shin, W. C., Koo, G., Kim, M. S., Lee, C., and Lee, S. J., "Automated defect inspection system for metal surfaces based on deep learning and data augmentation," *Journal of Manufacturing Systems* **55**, 317–324 (Apr. 2020).

[12] Valvano, G., Agostino, A., De Magistris, G., Graziano, A., and Veneri, G., "Controllable Image Synthesis of Industrial Data using Stable Diffusion," in [*2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*], 5342–5351, IEEE, Waikoloa, HI, USA (Jan. 2024).

[13] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I., "Zero-Shot Text-to-Image Generation,"

[14] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M., "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding," (2022). Version Number: 1.

[15] Wang, R., Schmedding, S., and Huber, M. F., "Improving the Effectiveness of Deep Generative Data," in [*2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*], 4910–4920, IEEE, Waikoloa, HI, USA (Jan. 2024).

[16] Paulin, G. and Ivasic-Kos, M., "Review and analysis of synthetic dataset generation methods and techniques for application in computer vision," *Artificial Intelligence Review* **56**, 9221–9265 (Sept. 2023).

[17] Dwibedi, D., Misra, I., and Hebert, M., "Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection," in [*2017 IEEE International Conference on Computer Vision (ICCV)*], 1310–1319, IEEE, Venice (Oct. 2017).

[18] Mery, D., "Aluminum Casting Inspection using Deep Object Detection Methods and Simulated Ellipsoidal Defects," *Machine Vision and Applications* **32**, 72 (May 2021).

[19] Li, C.-L., Sohn, K., Yoon, J., and Pfister, T., "CutPaste: Self-Supervised Learning for Anomaly Detection and Localization," in [*2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*], 9659–9669, IEEE, Nashville, TN, USA (June 2021).

[20] Mohammadzadeh, M., Kremer, G. E. O., Olafsson, S., and Kremer, P. A., "AI-Driven Crack Detection for Remanufacturing Cylinder Heads Using Deep Learning and Engineering-Informed Data Augmentation," *Automation* **5**, 578–596 (Nov. 2024).

[21] Rill-García, R., Dokladalova, E., and Dokládal, P., "Syncrack: Improving Pavement and Concrete Crack Detection through Synthetic Data Generation:," in [*Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*], 147–158, SCITEPRESS - Science and Technology Publications, Online Streaming, — Select a Country — (2022).

[22] Haselmann, M. and Gruber, D., "Supervised Machine Learning Based Surface Inspection by Synthetizing Artificial Defects," in [*2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*], 390–395, IEEE, Cancun, Mexico (Dec. 2017).

[23] Haselmann, M. and Gruber, D. P., "Pixel-Wise Defect Detection by CNNs without Manually Labeled Training Data," *Applied Artificial Intelligence* **33**, 548–566 (May 2019).

[24] Su, H., Qi, C. R., Li, Y., and Guibas, L. J., "Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views," in [*2015 IEEE International Conference on Computer Vision (ICCV)*], 2686–2694, IEEE, Santiago, Chile (Dec. 2015).

[25] Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., and Birchfield, S., "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)*], 1082–10828, IEEE, Salt Lake City, UT (June 2018).

[26] Schoepflin, D., Holst, D., Gomse, M., and Schüppstuhl, T., "Synthetic Training Data Generation for Visual Object Identification on Load Carriers," *Procedia CIRP* **104**, 1257–1262 (2021).

[27] Martinez-Gonzalez, P., Oprea, S., Garcia-Garcia, A., Jover-Alvarez, A., Orts-Escolano, S., and Garcia-Rodriguez, J., "UnrealROX: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation," *Virtual Reality* **24**, 271–288 (June 2020).

[28] Greff, K., Belletti, F., Beyer, L., Doersch, C., Du, Y., Duckworth, D., Fleet, D. J., Gnanapragasam, D., Golemo, F., Herrmann, C., Kipf, T., Kundu, A., Lagun, D., Laradji, I., Liu, H.-T., Meyer, H., Miao, Y., Nowrouzezahrai, D., Oztireli, C., Pot, E., Radwan, N., Rebain, D., Sabour, S., Sajjadi, M. S. M., Sela, M., Sitzmann, V., Stone, A., Sun, D., Vora, S., Wang, Z., Wu, T., Yi, K. M., Zhong, F., and Tagliasacchi, A., "Kubric: A scalable dataset generator," in [*2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*], 3739–3751, IEEE, New Orleans, LA, USA (June 2022).

[29] Denninger, M., Winkelbauer, D., Sundermeyer, M., Boerdijk, W., Knauer, M., Strobl, K. H., Humt, M., and Triebel, R., "BlenderProc2: A Procedural Pipeline for PhotorealisticRendering," *Journal of Open Source Software* **8**, 4901 (Feb. 2023).

[30] Moonen, S., Vanherle, B., De Hoog, J., Bourgana, T., Bey-Temsamani, A., and Michiels, N., "CAD2Render: A Modular Toolkit for GPU-accelerated Photorealistic Synthetic Data Generation for the Manufacturing Industry," in [*2023 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*], 583–592, IEEE, Waikoloa, HI, USA (Jan. 2023).

[31] Raistrick, A., Lipson, L., Ma, Z., Mei, L., Wang, M., Zuo, Y., Kayan, K., Wen, H., Han, B., Wang, Y., Newell, A., Law, H., Goyal, A., Yang, K., and Deng, J., "Infinite Photorealistic Worlds Using Procedural Generation," in [*2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*], 12630–12641, IEEE, Vancouver, BC, Canada (June 2023).

[32] Nikolenko, S. I., "Synthetic Data for Deep Learning," (Sept. 2019). arXiv:1909.11512 [cs].

[33] Movshovitz-Attias, Y., Kanade, T., and Sheikh, Y., "How Useful Is Photo-Realistic Rendering for Visual Learning?," in [*Computer Vision – ECCV 2016 Workshops*], Hua, G. and Jégou, H., eds., **9915**, 202–217, Springer International Publishing, Cham (2016). Series Title: Lecture Notes in Computer Science.

[34] Eversberg, L. and Lambrecht, J., "Generating Images with Physics-Based Rendering for an Industrial Object Detection Task: Realism versus Domain Randomization," *Sensors* **21**, 7901 (Nov. 2021).

[35] Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Smagt, P. V. D., Cremers, D., and Brox, T., "FlowNet: Learning Optical Flow with Convolutional Networks," in [*2015 IEEE International Conference on Computer Vision (ICCV)*], 2758–2766, IEEE, Santiago (Dec. 2015).

[36] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P., "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)*], 23–30 (Sept. 2017). ISSN: 2153-0866.

[37] Prakash, A., Boochoon, S., Brophy, M. A., Acuna, D., Cameracci, E., State, G., Shapira, O., and Birchfield, S., "Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data,"

[38] Boikov, A., Payor, V., Savelev, R., and Kolesnikov, A., "Synthetic Data Generation for Steel Defect Detection and Classification Using Deep Learning," *Symmetry* **13**, 1176 (June 2021).

[39] Cordier, A., Gutierrez, P., and Plessis, V., "Improving generalization with synthetic training data for deep learning based quality inspection," (Feb. 2022). arXiv:2202.12818 [cs].

[40] Gutierrez, P., Luschkova, M., Cordier, A., Shukor, M., Schappert, M., and Dahmen, T., "Synthetic training data generation for deep learning based quality inspection," in [*Fifteenth International Conference on Quality Control by Artificial Vision*], Komuro, T. and Shimizu, T., eds., 13, SPIE, Tokushima, Japan (July 2021).

[41] Lebert, D., Plouzeau, J., Farrugia, J.-P., Danglade, F., and Merienne, F., "Synthetic Data Generation for Surface Defect Detection," in [*Extended Reality*], De Paolis, L. T., Arpaia, P., and Sacco, M., eds., **13446**, 198–208, Springer Nature Switzerland, Cham (2022). Series Title: Lecture Notes in Computer Science.

[42] Hoskere, V., Narazaki, Y., and Spencer, B. F., "Physics-Based Graphics Models in 3D Synthetic Environments as Autonomous Vision-Based Inspection Testbeds," *Sensors* **22**, 532 (Jan. 2022).

[43] Bonfiglioli, L., Toschi, M., Silvestri, D., Fioraio, N., and De Gregorio, D., "The Eyecandies Dataset for Unsupervised Multimodal Anomaly Detection and Localization," in [*Computer Vision – ACCV 2022*], Wang, L., Gall, J., Chin, T.-J., Sato, I., and Chellappa, R., eds., **13845**, 459–475, Springer Nature Switzerland, Cham (2023). Series Title: Lecture Notes in Computer Science.

[44] Mohanty, S., Su, E., and Ho, C.-C., "Reinforcement learning optimized digital twin based synthetic data generation for defect detection of titanium spacer," in [*Automated Visual Inspection and Machine Vision V*], Beyerer, J. and Heizmann, M., eds., 6, SPIE, Munich, Germany (Aug. 2023).

[45] Mohanty, S., Su, E., and Ho, C.-C., "Enhancing titanium spacer defect detection through reinforcement learning-optimized digital twin and synthetic data generation," *Journal of Electronic Imaging* **33** (Jan. 2024).

[46] Effenberger, I., Seiler, F., and Eichinger, V., "Synthetische Daten für die Automatisierung mit KI/Synthetic data for AI-based automation," *wt Werkstattstechnik online* **114**(09), 490–496 (2024).

[47] Schorr, C., Hocke, S., Masiak, T., and Trampert, P., "A Scalable Synthetic Data Creation Pipeline for AI-Based Automated Optical Quality Control:," in [*Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*], 37–46, SCITEPRESS - Science and Technology Publications, Dijon, France (2024).

[48] Singh, A. R., Bashford-Rogers, T., Hazra, S., and Debattista, K., "Generating Synthetic Training Images to Detect Split Defects in Stamped Components," *IEEE Transactions on Industrial Informatics* **20**, 4816–4827 (Mar. 2024).

[49] Martinet, A., Galin, E., Desbenoit, B., and Akkouche, S., "Procedural modeling of cracks and fractures," in [*Proceedings Shape Modeling Applications, 2004.*], 346–349, IEEE, Genova, Italy (2004).

[50] Retzlaff, M.-G., Hanika, J., Dachsbacher, C., and Beyerer, J., "Potential and Challenges of using Computer Graphics for the Simulation of Optical Measurement Systems," in [*Tagungsband*], 322–329, AMA Service GmbH, Von-Münchhausen-Str. 49, 31515 Wunstorf, Germany, Nürnberg, Germany (2016).

[51] Retzlaff, M.-G., Richter, M., and Langle, T., "Combining synthetic image acquisition and machine learning: accelerated design and deployment of sorting systems," (2016).

[52] Retzlaff, M.-G., Stabenow, J., Beyerer, J., and Dachsbacher, C., "Synthesizing images using parameterized models for automated optical inspection (AOI)," *tm - Technisches Messen* **82**, 251–261 (May 2015).

[53] Dahmen, T., Trampert, P., Boughorbel, F., Sprenger, J., Klusch, M., Fischer, K., Kübel, C., and Slusallek, P., "Digital reality: a model-based approach to supervised learning from synthetic data," *AI Perspectives* **1**, 2 (Dec. 2019).

[54] Bosnar, L., Saric, D., Dutta, S., Weibel, T., Rauhut, M., Hagen, H., and Gospodnetic, P., "Image Synthesis Pipeline for Surface Inspection," (Nov. 2020).

[55] Bosnar, L., Rauhut, M., Hagen, H., and Gospodnetic, P., "Texture Synthesis for Surface Inspection," in [*LEVIA'22 : Leipzig Symposium on Visualization in Applications 2022*], Leipzig University (July 2022).

[56] Schmedemann, O., Baaß, M., Schoepflin, D., and Schüppstuhl, T., "Procedural synthetic training data generation for AI-based defect detection in industrial surface inspection," *Procedia CIRP* **107**, 1101–1106 (2022).

[57] Fulir, J., Bosnar, L., Hagen, H., and Gospodnetić, P., "Synthetic Data for Defect Segmentation on Complex Metal Surfaces," in [*2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*], 4424–4434, IEEE, Vancouver, BC, Canada (June 2023).

[58] Fulir, J., Jeziorski, N., Bosnar, L., Hagen, H., Redenbach, C., Gospodnetić, P., Herrfurth, T., Trost, M., and Gischkat, T., "SYNOSIS: Image synthesis pipeline for machine vision in metal surface inspection," (Oct. 2024). arXiv:2410.14844 [cs].

[59] Cook, R. L., "Shade trees," *ACM SIGGRAPH Computer Graphics* **18**, 223–231 (July 1984).

[60] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P., "Microsoft COCO: Common Objects in Context," (Feb. 2015). arXiv:1405.0312 [cs].

[61] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A., "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision* **88**, 303–338 (June 2010).

[62] Li, X., Wang, W., Sun, L., Hu, B., Zhu, L., and Zhang, J., "Deep learning-based defects detection of certain aero-engine blades and vanes with DDSC-YOLOv5s," *Scientific Reports* **12**, 13067 (July 2022).

[63] Shang, H., Wu, J., Sun, C., Liu, J., Chen, X., and Yan, R., "Global Prior Transformer Network in Intelligent Borescope Inspection for Surface Damage Detection of Aeroengine Blade," *IEEE Transactions on Industrial Informatics* **19**, 8865–8877 (Aug. 2023).

[64] Ren, S., He, K., Girshick, R., and Sun, J., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," (Jan. 2016). arXiv:1506.01497 [cs].

[65] Hu, Y., He, C., Deschaintre, V., Dorsey, J., and Rushmeier, H., "An Inverse Procedural Modeling Pipeline for SVBRDF Maps," *ACM Transactions on Graphics* **41**, 1–17 (Apr. 2022).