Structural Health Monitoring
1–31
© The Author(s) 2025
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/14759217251340416
journals.sagepub.com/home/shm
Sage

Deep learning-based 3D image reconstruction and damage mapping using neural radiance fields (Nerfacto)

Geontae Kim and Youngjin Cha

Abstract
In structural health monitoring using computer vision, deep learning-based damage identification and three-dimensional (3D) reconstruction of the structure are current hot topics. Traditional photogrammetry techniques are cost-inefficient and time-consuming for 3D reconstruction, and there is no such solid 3D pixelwise damage mapping technique. To overcome these limitations, a new deep neural network (DNN)-based 3D reconstruction method, including damage mapping, is proposed in this article. As the DNN-based 3D reconstruction method, Nerfacto—an advanced version of Neural Radiance Fields models—was selected for achieving high-fidelity 3D reconstruction. This Nerfacto model was modified to create a high-definition 3D reconstruction model of the structure of interest (i.e., a 3-span bridge system). To map damages within the reconstructed 3D model using the modified Nerfacto, the state-of-the-art semantic transformer representation network (STRNet) with test time augmentation (TTA) was also developed for precise pixel-wise crack segmentation. Through extensive case studies, including parametric studies, we found that the modified Nerfacto can learn various appearance features of the structure and generate a very high-definition 3D model. Moreover, the segmented damage (i.e., cracks) from the STRNet with TTA could be mapped onto the reconstructed 3D model. This study demonstrates the potential of combining deep learning with 3D reconstruction for proactive and preventative maintenance strategies, ensuring the safety and longevity of vital structural assets.

Keywords
SHM, damage detection, deep learning, crack segmentation, 3D reconstruction, neural radiance fields, NeRF, Nerfacto, digital twin

Introduction

Structural health monitoring (SHM) is a critical process for assessing the condition and predicting the lifespan of infrastructural assets.[1] It plays a vital role in ensuring the safety and longevity of structures while aiding in cost-effective maintenance. The importance of SHM worldwide is highlighted by the continuous aging of bridge infrastructure. For example, in Canada, since the 1970s, there has been a noticeable increase in the average age of bridges, indicating a consistent aging process exacerbated by inadequate investment.[2] By 2003, many federal and provincial bridges had surpassed half of their expected 46-year lifespan. Federal bridges, though only 3% of the total, averaged 26.4 years old, while provincial bridges, which make up 57% of the total, saw their average age rise from 15.4 to 24.6 years from 1963 to 2003, largely due to low provincial investment. Municipal bridges, younger on average at 19.0 years, comprised 39% of the infrastructure.[3] This aging trend, especially among provincial bridges, shows the need for regular inspections to maintain safety and prolong their usable life. Regularly conducted SHM not only ensures structural safety but also contributes to reducing maintenance costs, making it an indispensable practice in the field of civil engineering and infrastructure management.[4]

While SHM systems have traditionally relied on vibration sensors,[5] these methods face challenges, particularly when applied to large-scale civil infrastructures. Techniques that utilize numerical methods, as

Department of Civil Engineering, University of Manitoba, Winnipeg, MB, Canada

Corresponding author:
Youngjin Cha, Department of Civil Engineering, University of Manitoba, SP427 Stanley Pauley Engineering, 105 Dafoe Rd W, Winnipeg, MB, R3T 6B3, Canada.
Email: young.cha@umanitoba.ca

explored in various studies, often struggle with the effects of uncertainty and non-uniform environmental influences.[6] A major hurdle in these approaches is the difficulty in distinguishing between actual structural damage, malfunctions in the sensory system, and noisy signals. This complexity necessitates on-site verification of the sensing systems and the structures themselves, which requires qualified technicians to visit the structure, resulting in an unnecessary use of time and financial resources.

Among various indicators of structural deficiencies, cracks are particularly noteworthy. Commonly arising from factors such as cyclical loading, drying shrinkage, thermal contraction and expansion, and overall structural deterioration, cracks are critical markers that demand attention.[7] Addressing these damages promptly is essential for ensuring the long-term durability of structures and preventing the propagation of cracking. Effective monitoring and sealing of cracks are therefore pivotal in maintaining structural integrity and extending the lifespan of infrastructural assets.[8]

The evolution of vision-based equipment and image-processing technologies has advanced the field of SHM. Computer vision has emerged as an efficient tool for damage identification, offering a more reliable alternative to the traditional visual evaluations conducted by human experts, which are often labor-intensive and subject to variability.[9,10]

Traditional image-based techniques for crack segmentation, detection, and classification have been extensively researched.[11] These methods, including thresholding-based, edge-based, and data-driven approaches, have gradually become less favored due to limitations in accuracy and difficulties in crack segmentation in complex backgrounds.[12] In contrast, the advent of deep learning, particularly deep convolutional neural networks (CNNs), has revolutionized damage detection. Deep CNNs excel in extracting multilevel robust features from images and continuously refining internal parameters for automatic recognition.[13–15]

A notable development in this domain is the U-net architecture, initially proposed for medical image analysis,[16] which has been effectively adapted for both medical and non-medical image segmentation, including crack detection.[17,18] Further advancements have been made; Fu et al.[19] also attempted to integrate Deeplabv3 + and Xception for bridge crack segmentation. Despite these developments, challenges remain, particularly in applying U-net to complex scenes with diverse visual elements. Addressing this, an object-specific network targeting structural cracks, namely SDDNet,[20] was developed using various advanced modules (DenSep module and Atrous Spatial Pyramid Pooling (ASPP)) with advanced operations such as depthwise convolution and dilated convolutions. Another object-specific network, semantic transformer representation network (STRNet),[21] was proposed by designing an STR module that has three different configurations that are regularly repeated with different filters and activation functions, including attention modules in the encoder and decoder for crack segmentation. This offers real-time processing of normal video with a large input frame size of $1200 \times 800 \times 3$, achieving state-of-the-art (SOTA) performance with a mean intersection over union (mIoU) of 0.93 in varied and complex scene environments. These advances in computer vision and deep learning emphasize the dynamic nature of SHM and underscore the shift from traditional methods to more sophisticated and automated techniques that increase the accuracy and efficiency of structural damage detection.

A critical challenge in applying CNNs to damage detection in real structures is the requirement for extensive training samples.[22] Capturing a comprehensive range of damage scenarios for in-service structures to train these models is often difficult, if not impossible. This limitation can restrict the practical application of CNNs in real-world SHM.[23] To address this issue, researchers have explored methods like transfer learning and semi-supervised learning. Transfer learning, as discussed by Teng et al.,[24] involves adapting a pre-trained model to a new but related problem, while semi-supervised learning, as described by Xiang et al.,[25] leverages a combination of labeled and unlabeled data for training. However, both approaches require re-training, which can be time-consuming and resource-intensive. Additionally, they often necessitate larger datasets.

In the context of SHM, where time efficiency is paramount for rapid monitoring and assessment,[26] test time augmentation (TTA) emerges as a valuable method. TTA involves applying various augmentations to the test images and then averaging or combining the predictions to improve the model's performance.[27] This technique does not require retraining the model; instead, it enhances the prediction accuracy at the time of testing by leveraging diverse variations of the input data. As highlighted by Wang et al.,[28] TTA operates similarly to an ensemble approach, combining results from different augmentations to achieve a more robust and accurate outcome. This makes TTA particularly suited for SHM applications where quick and reliable damage assessment is crucial.

The detection of damage in a single image in SHM, while useful, has limitations, especially when dealing with large structures. A single image typically covers only a small portion of an entire structure, making it challenging to contextualize the extent of the damage[29] and to localize the damage even though some articles have used geo-tagging for 2D damage localization.[15,30,31] Therefore, accurately mapping detected damages from 2D images to 3D world coordinates is crucial, particularly for large structures like bridges. This process enables a more comprehensive understanding of the damage distribution and severity.

Recently, 3D reconstruction as a digital twin of structures has emerged as a transformative solution for intelligent SHM. It not only enhances the accuracy of damage assessment but also paves the way for more intelligent and predictive maintenance strategies.[32,33] The 3D models generated for SHM purposes should contain enriched surface texture with high spatial resolution for damage assessment.[34] This involves creating a 3D representation of an object, where data points can be acquired through photogrammetric methods or laser scanning. In computer vision, various techniques are implemented using multiple images, as opposed to relying solely on stereo images.[35] One such technique is Structure from Motion (SfM), which has been effectively used to create detailed 3D models of objects and landscapes.[35,36] For instance, Xue et al.[37] utilized an SfM-based deep learning method to assess defects in tunnels, demonstrating the method's applicability in monitoring expansive areas.

Photogrammetry has also been utilized to project detected concrete cracks onto reconstructed 3D models, with CNN-based methods like UNet enhancing the accuracy of these projections.[38] However, photogrammetry-based 3D reconstruction presents its challenges. It can be time-consuming and computationally intensive.[39] Moreover, reconstructing planar surfaces solely through photogrammetry poses specific difficulties, so lasers can be used to overcome these,[40] necessitating the exploration of more efficient and effective methods in SHM.

In the realm of recent 3D reconstruction, neural radiance fields (NeRF), introduced by Mildenhall et al.,[41] have emerged as a breakthrough approach, attracting researchers' attention as a viable alternative to traditional photogrammetry. This method distinguishes itself from photogrammetry by effectively scanning objects with sparse feature points, thus offering a more flexible solution for 3D reconstruction. Departing from the reliance on manual texture mapping and geometric primitives such as triangles and voxels, characteristic of traditional methods, NeRF employs a neural network to directly model the visual characteristics of a scene. It learns the scenes from images collected from different camera locations. This novel approach facilitates capturing complex lighting effects, including reflections and transparencies, by creating unique representations of points from varying viewing angles. As a result, NeRF achieves a more detailed and accurate rendering of scenes,[42] presenting a potential method for use in SHM.

Recent research has highlighted the potential of utilizing NeRF for advanced applications in structural health monitoring and construction automation. Pal et al.[43] integrated NeRF with building information modeling (BIM) systems to visually monitor the progress of construction activities, demonstrating NeRF's ability to support project management by providing precise visual documentation of construction stages. Additionally, Yu et al.[44] applied NeRF to 3D damage mapping on underwater bridge piers, showcasing its effectiveness in capturing detailed representations of structural damages. However, these applications typically focus on specific parts or limited areas, showing particular viewpoints or sections of structures.

This limited scope is partly due to NeRF's inherent limitations in generating 3D reconstructed models: (1) it requires significant computational resources, therefore it only works for very small-sized objects such as chairs or desks; (2) it has limited performance in predicting the actual pixel opacity and RGB values due to poor generalization performance; and (3) it is very sensitive to hyperparameters, making it unreliable for large-scale civil infrastructures.

To address the limitations of the original NeRF, particularly in large-scale environments, Tancik et al.[45] introduced the prototype Nerfacto model, an advanced variant designed to enhance 3D scene reconstruction capabilities. This model tackles computational challenges by employing a smaller neural network architecture in conjunction with multiresolution hash encoding and proposal sampling techniques, thereby reducing computational demands. Additionally, the Nerfacto model enhances the accuracy of RGB value predictions through advanced methods such as proposal sampling, appearance embedding, and spherical harmonics encoding. However, we find that this prototype Nerfacto still faces limitations when applied to civil infrastructure.[46]

To overcome these challenges, this article proposes an enhanced approach to Nerfacto-based 3D damage mapping for structures. The proposed method incorporates advanced deep learning-based pixelwise crack segmentation using state-of-the-art (SOTA) models, STRNet and TTA. This article is composed of section "Methodology," section "Case studies," section "Conclusions," and section "References."

Methodology

In this article, we propose a new 3D damage mapping method by integrating an advanced deep learning-based pixelwise damage segmentation method with a 3D image reconstruction method. The overall process of the proposed method is illustrated in Figure 1. As

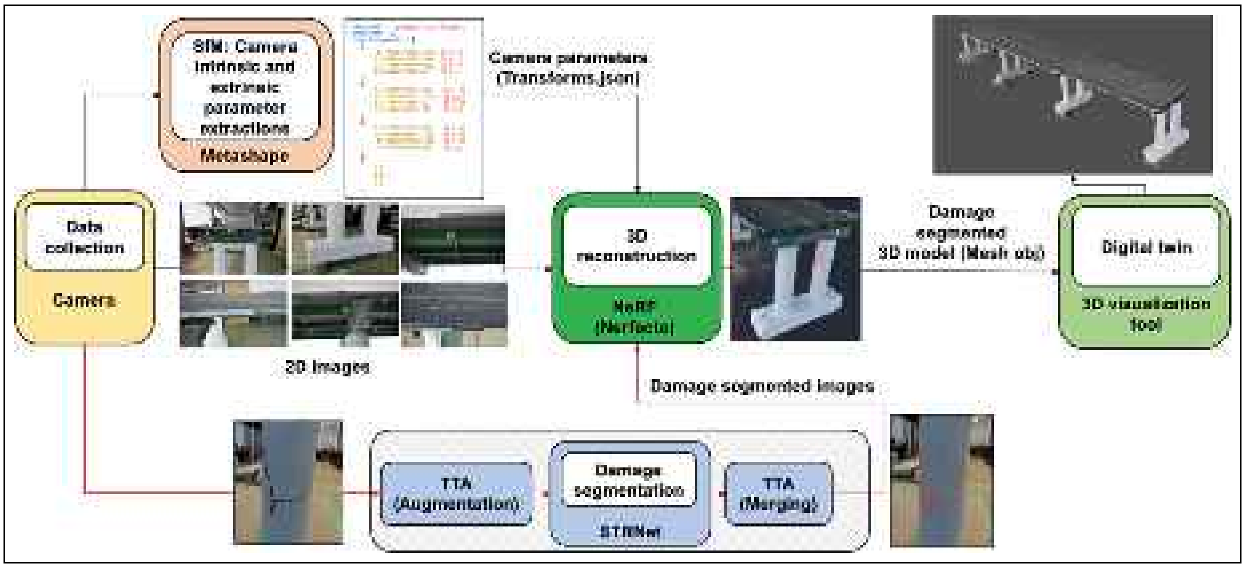


Figure 1. Flowchart of the proposed method framework.

the initial step, 2D RGB images of a structure of interest are collected. For 3D reconstruction of the structure, SfM is used to extract camera parameters using the Agisoft Metashape software tool.[47] The calculated camera parameters are then forwarded to the Nerfacto model,[45] which is an advanced NeRF-based model. Additionally, the collected 2D RGB images are fed into the newly proposed TTA-STRNet for pixelwise damage segmentation. The 2D images with segmented damage are subsequently input into the Nerfacto model for 3D rendering and 3D damage mapping using the calculated camera parameters. Ultimately, the reconstructed 3D damage map can be displayed using any 3D visualization tool as comprehensive damage information. All the details of each step are described in the following subsections.

Camera parameter extraction

To employ the NeRF-based Nerfacto model in our methodology for 3D reconstruction, with segmented damage or without segmented damage, it is essential to obtain each image's intrinsic and extrinsic camera parameters, represented by the matrices $\boldsymbol{K}$ (Equation (1)) and $\boldsymbol{R}|\boldsymbol{t}$ (Equation (2)), respectively.

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

where f_x and f_y (typically the same) are focal lengths, $[c_x, c_y]$ is the principal point representing the image sensor size of the camera used, and s (generally 0 in modern camera) represents a camera sensor lens distortion.

$$\boldsymbol{R}|\boldsymbol{t} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

where $r_{i,j}$ represents the transformation matrix from the 3D global coordinate system (GCS) $(X_G, Y_G,\ Z_G)$ to the local coordinate system (LCS) $(X_c,\ Y_c,\ Z_c)$ of the camera's position. $\boldsymbol{t} = [t_x,\ t_y, t_z]^T$ is the camera's position vector in the GCS.

These parameters ($\boldsymbol{K}$, and $\boldsymbol{R}|\boldsymbol{t}$), along with each image, are initially fed into the Nerfacto model, as shown in Figure 2 illustrates the relationships between GCS $(X_G, Y_G,\ Z_G)$ and each image's LCS $(X_c,\ Y_c,\ Z_c)$. For 3D reconstruction, many images taken from various locations and angles are required, with each image having its own LCS.

As shown in Figure 2, when an arbitrary 3D point $[x_c,\ y_c,\ z_c]^T$ of the object (i.e., bridge) in the LCS is z_c distance away from the camera's focus, the relationship between the normalized image coordinates $(x_n,\ y_n)$ and the arbitrary 3D point is expressed by Equation (3). This normalized image plane is defined at a unit length from the camera focus.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = z_c \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} \tag{3}$$

These normalized image coordinates $[x_n,\ y_n,\ 1]^T$ are then converted into pixel coordinates where the camera

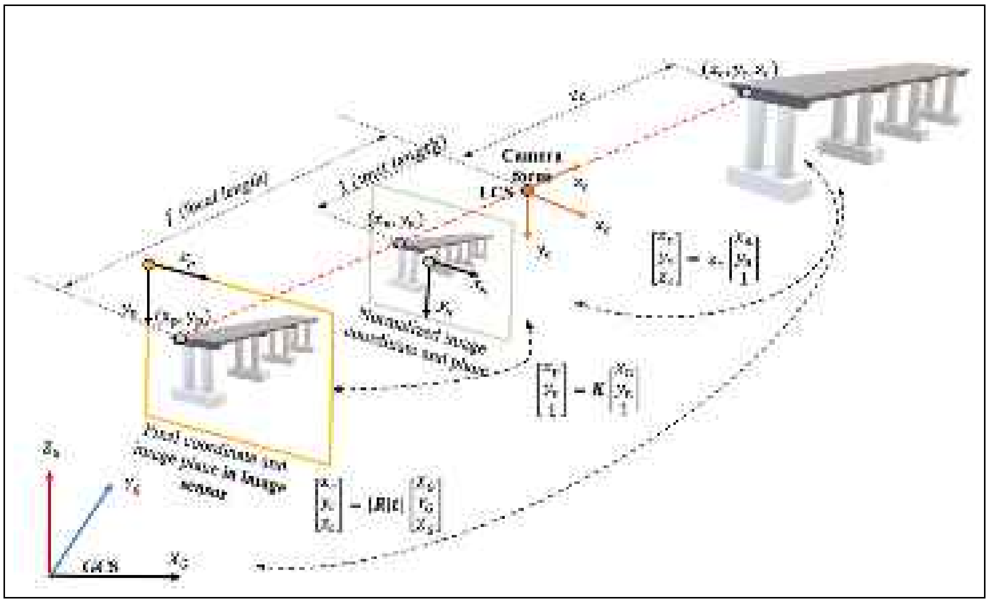

Camera form
LCS
GCS
f (focal length)
1 (unit length)
Normalized image coordinate and plane
Pixel coordinate and image plane in image sensor

Figure 2. Camera projection model.

image sensor is located, using the intrinsic camera matrix $\boldsymbol{K}$ as shown in Equation (4).

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} \tag{4}$$

Consequently, Equation (5) explains how the 3D spatial coordinates $[x_c,\ y_c,\ z_c]^T$ can be converted to the pixel coordinates.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = z_c \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = z_c \boldsymbol{K}^{-1} \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} \tag{5}$$

When images of an object are captured from varying angles and positions, the LCS for each image differs. By applying the extrinsic matrix $\boldsymbol{R}|\boldsymbol{t}$, as described in Equation (6) in the context of computer vision and graphics, it defines the transformation from GCS to LCS and can work in reverse with the inverse of this matrix.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = [\boldsymbol{R}|\boldsymbol{t}] \begin{bmatrix} X_G \\ Y_G \\ Z_G \\ 1 \end{bmatrix} \tag{6}$$

The first 3×3 rotation matrix $\boldsymbol{R}$ is responsible for rotating coordinates from world space to the camera space. Each element r_{ij} represents the cosine of the angle between the GCS and the LCS. The complete rotation matrix $\boldsymbol{R}$ is computed by multiplying the individual rotation matrices as follows Equation (7):

$$\boldsymbol{R} = \boldsymbol{R}_z(\boldsymbol{\theta}_z) \times \boldsymbol{R}_y(\boldsymbol{\theta}_y) \times \boldsymbol{R}_x(\boldsymbol{\theta}_x), \tag{7}$$

where each $\boldsymbol{R}_x(\theta_x)$, $\boldsymbol{R}_y(\theta_y)$, and $\boldsymbol{R}_z(\theta_z)$ are defined as

$$\boldsymbol{R}_x(\boldsymbol{\theta}_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{bmatrix}$$

$$\boldsymbol{R}_y(\boldsymbol{\theta}_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix},$$

$$\text{and } \boldsymbol{R}_z(\boldsymbol{\theta}_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ respectively.}$$

To obtain the extrinsic camera matrix $\boldsymbol{R}|\boldsymbol{t}$, we turn to SfM, a technique widely used in the field of computer vision. Although several SfM tools are available, Metashape is chosen for its robustness and accuracy in extracting camera parameters. As a result from the SfM method using this software tool, a file named "Transforms.json" is created, containing both intrinsic parameters and extrinsic matrices. This file is crucial for Nerfacto applications, ensuring accurate modeling of the camera's viewpoint for ray projection and color value extraction in 3D reconstruction. The intrinsic parameters can be extracted based on the image size $(H,\ W)$ and the focal length $(f_x,\ f_y)$, which are typically stored within the image files. The principal point $(c_x,\ c_y)$, generally located at the center of the image sensor, can be estimated as half the size of H and W. For this reason, while the intrinsic matrix $\boldsymbol{K}$ is essential,

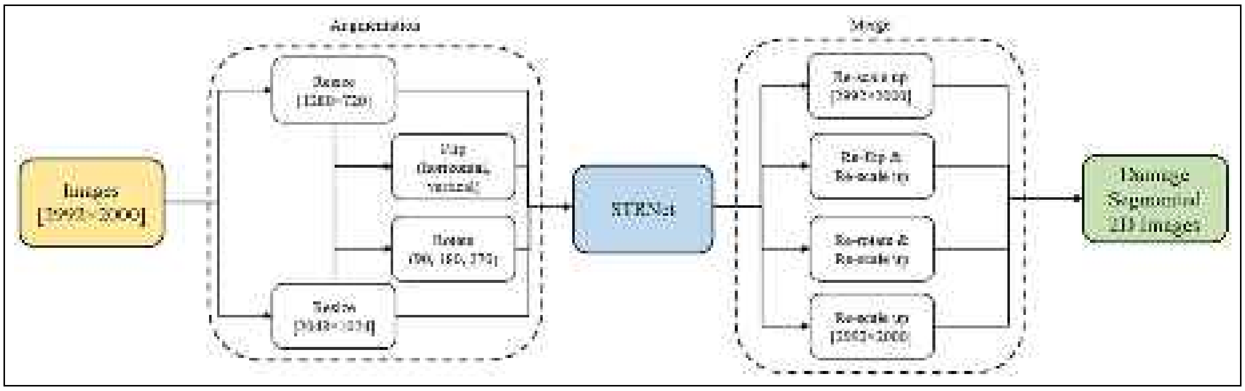


Figure 3. STRNet + TTA structure.
STRNet: semantic transformer representation network; TTA: test time augmentation.

the primary content of the file involves the extrinsic parameters—hence the name "transforms." However, the file still includes values for the focal lengths (f_x, f_y) and the principal point (c_x, c_y), simplifying their use in Nerfacto application.

TTA-STRNet crack segmentation

In this article, the main contribution is the development of 3D damage mapping, therefore, for the crack damage segmentation method, we utilized STRNet[21] for segmenting structural damage within 2D RGB images. STRNet, a sophisticated deep learning network, is tailored for real-time, pixel-level crack segmentation against complex backgrounds with large input image frames (1280 × 720). STRNet achieved a mIoU of 92.6% and a processing speed of 49 frames per second for the same image size, making it a state-of-the-art method in this field.

However, in this article, we use a specific bridge system that is in the lab, therefore, it is quite challenging to have various different testing images for the STRNet. To overcome this challenge, we combined TTA with STRNet as a solution. TTA is an image augmentation technique that can be used in the cases where there is only a limited number of testing images. This method can augment the limited testing images for the STRNet.

Test time augmentation. TTA is a methodology implemented during the testing phase, unlike standard data augmentation, which is applied during the training phase. TTA involves applying a series of image transformations, such as resizing, flipping, and rotating, during inference, and then aggregating the results to form a final prediction.

In our approach, as illustrated in Figure 3, images captured at a resolution of 2992 × 2000 pixels are used for 3D image reconstruction to achieve high-resolution results. However, for damage segmentation, the images are resized to 1280 × 720 and 2048 × 1024 because STRNet was originally developed as a real-time method for processing standard video frames with a size of 1280 × 720. After detecting damage, the images are rescaled back to their original dimensions (2992 × 2000), as shown in Figure 3. To further augment our data and enhance the model's robustness, we apply horizontal and vertical flips as well as rotations (90°, 180°, and 270°). This augmentation strategy results in 12 distinct test cases, allowing us to evaluate the model's performance across various image sizes and material characteristics.

Integrating TTA with STRNet not only enhances the model's performance without requiring retraining but also minimizes bias by analyzing multiple augmented versions of the same image during inference. This method improves the deep learning model's accuracy and generalizability. By implementing various augmentations at test time and combining the outcomes, TTA functions in a manner akin to an ensemble method, ensuring prompt and accurate damage assessment essential for effective SHM.[28] The details of STRNet and TTA are elaborated upon in subsequent sections.

The semantic transformer representation network. The architecture of STRNet is specifically designed for crack segmentation, featuring an attention-based encoder and a multi-head attention-based decoder incorporated within the STR module, which is configured in three different ways and repeated irregularly 11 times. The inclusion of coarse up-sampling, a focal-Tversky loss function, and a learnable swish activation function simplifies the network's design without compromising its rapid processing capabilities. Figure 4 depicts the complete STRNet architecture, starting with the input

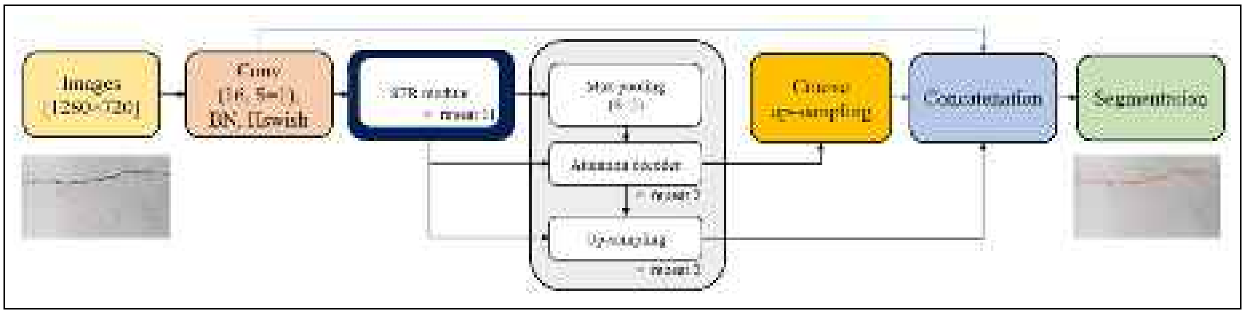


Figure 4. The STRNet architecture.[21]
STRNet: semantic transformer representation network.

images being processed initially in the first convolution block.

This block integrates convolution with the Hswish activation function[48] and batch normalization (BN).[49] The feature map produced by this convolutional block is then fed into the STR module, a process that repeats 11 times. After passing through the STR module, the output undergoes max pooling and is then input into the attention decoder. Utilizing skipped connections to retain multi-dimensional features, this decoder processes the data in two iterations. It then performs two types of up-sampling, utilizing both standard and coarse up-sampling techniques to return the data to the input image's original dimensions. The standard up-sampling step is repeated twice, reactivating the skipped connections.

In the final step, the outputs from the up-sampling block, the initial convolution block, and the coarse up-sampling block are combined. This combined output is then subjected to pointwise convolution (PW). This convolution step yields the final output, enabling precise pixel-level segmentation of cracks. More details can be found in Kang and Cha.[21]

NeRF for 3D reconstruction

The motivation for employing 3D reconstruction in our research stems from its capability to provide more intuitive visualizations, precise localization, and quantification of damage compared to traditional 2D methods. Additionally, 3D reconstructions can be exported as mesh OBJ files, which are beneficial for various downstream applications within the digital twin framework, such as 3D simulation and finite element analysis (FEA).

The original NeRF model, developed by Mildenhall et al.,[41] synthesizes 3D models from 2D images by creating a scene with continuous volumetric functions encoded into a deep neural network. This network is comprised of a deep neural network (DNN) with fully connected layers. In the NeRF model, each point in space is assigned a density value that indicates its level of transparency or opacity and includes a view-dependent color that changes based on the viewing angle and distance. This deep neural network learns the RGB values and opacity of each pixel from the collected 2D image data through the training process. As a result, a well-trained NeRF model can generate highly detailed and photorealistic renderings from new viewpoints.

However, the original NeRF model exhibits certain limitations in terms of computational efficiency, memory usage, training speed, and rendering quality under different lighting conditions. To address these limitations, NeRF technology has evolved to include several specialized models, each designed to tackle specific challenges and enhance aspects of 3D reconstruction. For example, Mip-NeRF 360[50] utilized a proposal sampling method that iteratively refines 3D sample points to focus on areas with higher scene content, improving sampling efficiency. Instant-NGP[51] enhances training speed and computational efficiency through hash multi-resolution encoding. Another model, Ref-NeRF,[52] employs spherical harmonic encoding to efficiently capture directional information and realistically render complex lighting environments. Moreover, NeRF-W[53] introduces appearance embeddings to handle variations in training data conditions, such as different lighting setups. In this paper, we adopt the Nerfacto model,[45] which combines elements of Instant-NGP, Ref-NeRF, and NeRF-W. This integration enables the Nerfacto model to strike a balance between rapid training times and high-quality outputs.

Subsequent subsections will delve into the structure of the original NeRF model and the Nerfacto model in more detail, discussing how the coordinates of 3D points and the directions of rays for each image are determined using the camera intrinsic and extrinsic parameters discussed in section "Camera parameter extraction."

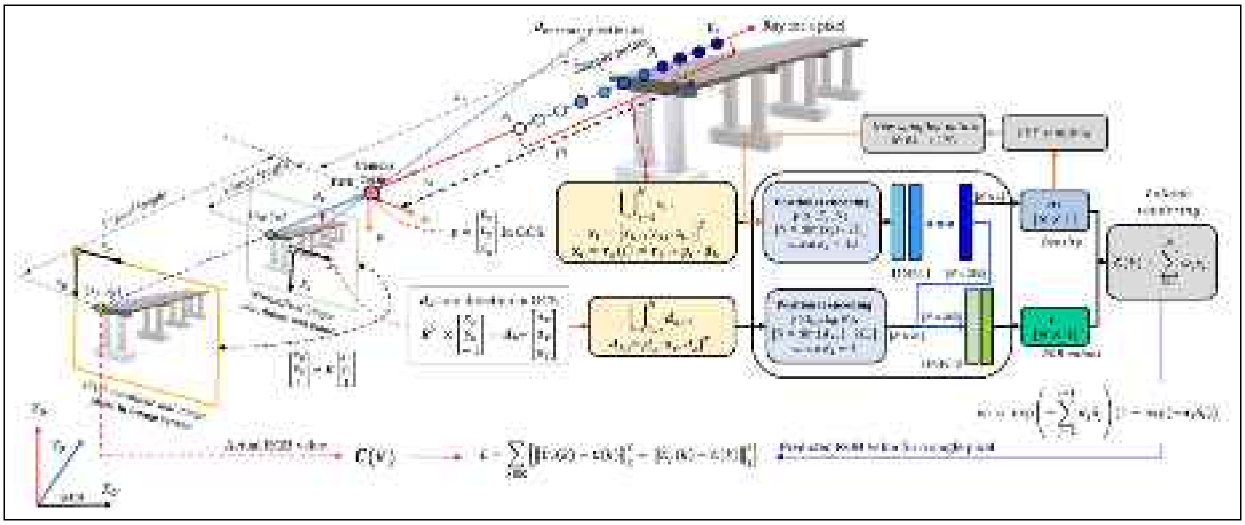


Figure 5. Overview of the NeRF model.
NeRF: neural radiance fields.

Neural radiance fields. As mentioned earlier, the original NeRF model has considerable limitations. Therefore, in this paper, we implemented the enhanced Nerfacto model. The overall process of Nerfacto is quite similar to that of NeRF. Therefore, we will first explain the general workflow of the NeRF model in this section, and then we will delve into the detailed workflow of Nerfacto based on NeRF's approach. In NeRF, initially, RGB images of our bridge model or crack damage segmented RGB images from STRNet, along with a "Transforms.json" file (see section "Camera parameter extraction"), are loaded into the NeRF model. The NeRF model is trained on each pixel of each image consecutively for the entire image, as typical neural networks are, through the concept of backpropagation.

As described in section "Camera parameter extraction," each image is transformed from the LCS to the GCS using $\boldsymbol{K}$ and $\boldsymbol{R}|\boldsymbol{t}$. In the image, each pixel has its own "ray" composed of a series of sample points ($\boldsymbol{x}_i$) through its corresponding directional vector ($\boldsymbol{d}_k$) in GCS, as shown in Figure 5. The number of samples for NeRF is N. The data from these N sample points are batched together, an operation denoted by $\bigcup$. These sample points are not real points that are measured but are artificially generated for the purpose of learning pixel colors at different distances and angles between the camera focus and the object, which is the unique technical novelty of NeRF and Nerfacto.

Using these various sample points, which have different distances between the camera and the object and are in the same line of the specific ray for each pixel, the NeRF model, which has DNN-I and -II, can learn the actual color in terms of RGB of the specific pixel through various collected images that have different fields of view, directions, and distances between the camera and the object of interest in GCS. To improve the model's accuracy, new 3D sample points are created during the DNN-I process by employing density-weighted probabilities via probability density function (PDF) sampling. These newly generated sample points are subsequently utilized to retrain the model, aiming to achieve a more precise RGB value for the specified pixel. The initial and subsequent phases are referred to as a coarse process and a fine process using PDF sampling, respectively.

Therefore, through a simple cost function, the predicted RGB values from the coarse and fine processes of NeRF will be compared with the actual RGB values coming from the collected image. Following that, through backpropagation, the DNNs will be trained to predict RGB values for different distances and directions for a more accurate establishment of the 3D model for the structures of interest. The detailed steps for the NeRF will be explained in the subsequent sections.

Extraction of sample points coordinates and directional vectors in GCS. Within the model, the 2D pixel coordinates $(x_p,\ y_p)$ as described in Equation (5) and depicted in Figures 2 and 5, for each pixel on the image sensor (i.e., the image) are transformed into normalized image coordinates $(x_n,\ y_n)$ using matrix $\boldsymbol{K}$ from Equation (1), as illustrated in Figure 5. This matrix includes intrinsic parameters such as focal length (f) and the principal point $(c_x,\ c_y)$. This transformation from pixel coordinates to normalized image coordinates is necessary due to their differing coordinate systems and to ascertain

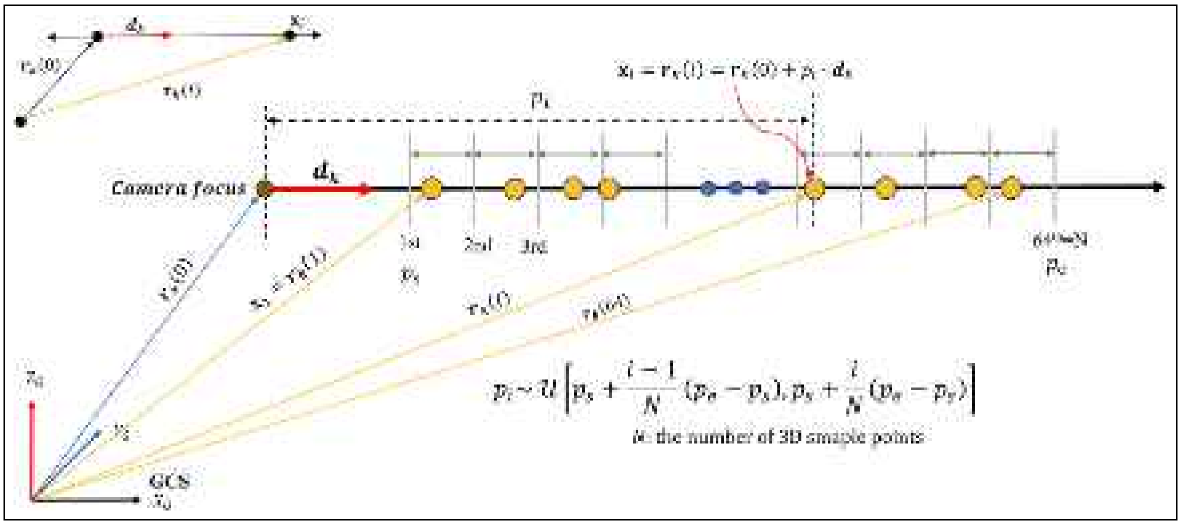

$x_i = r_k(i) = r_k(0) + p_i \cdot d_k$
p_i
Camera focus
d_k
1st
2nd
3rd
64th=N
p_s
p_e
$r_k(0)$
$x_1 = r_k(1)$
$r_k(i)$
$r_k(64)$
$p_i \sim \mathcal{U}\left[p_s + \frac{i-1}{N}(p_e - p_s), p_s + \frac{i}{N}(p_e - p_s)\right]$
N: the number of 3D smaple points
GCS

Figure 6. Stratified sampling along the ray.

the direction of the ray for each pixel in each image. Consequently, a unit directional vector $[x_n,\ y_n,\ -1]^T$ in the LCS is established. These normalized coordinates $(x_n,\ y_n)$ are calculated using the formulas $x_n = \frac{x_p - c_x}{f}$ and $y_n = \frac{y_p - c_y}{f}$, and are positioned at $(Z_c = -1)$ in the camera coordinate system, as elucidated with Figures 2 and 5. The choice of "-1" for (Z_c) instead of 1 is made because the normalized image plane is located at $(Z_c = -1)$ from the camera focus.

The unit directional vector $[x_n,\ y_n,\ -1]^T$ for each pixel in each image is in the LCS; therefore, it must be converted to the GCS through multiplication by the (3×3) transposed rotation matrix $(\boldsymbol{R}^T)$, which is derived from the (4×4) extrinsic matrices $(\boldsymbol{R}|\boldsymbol{t})$ in Equation (2), as detailed in Equation (8). By applying Equation (8), the ray directions for all distinct pixels can be determined. Here, $\boldsymbol{d_k}$ represents the unit directional vector for the kth pixel in the normalized image coordinates within the GCS.

$$\boldsymbol{R}^T \times \begin{bmatrix} x_n \\ y_n \\ -1 \end{bmatrix} = \boldsymbol{d_k} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}, \tag{8}$$

The subsequent step involves deriving 3D coordinates for the sample points situated along a particular ray in the GCS. NeRF presets the number of sample points to $(N = 64)$; consequently, for each pixel, there are 64 sample points distributed along the trajectory of each ray, as depicted in Figure 6.

Thus, within each ray's path, the sample points are uniformly allocated according to Equation (9):

$$p_i \sim \mathcal{U}\left[p_s + \frac{i-1}{N}(p_e - p_s),\ p_s + \frac{i}{N}(p_e - p_s)\right], \tag{9}$$

where (p_i) denotes the ith sampled distance along the ray, and $(\mathcal{U})$ signifies a uniform random selection within the interval $[p_s + \frac{i-1}{N}(p_e - p_s),\ p_s + \frac{i}{N}(p_e - p_s)]$, indicating that (p_i) is randomly chosen within this range. Here, (p_s) and (p_e) are the start and end sample points, respectively, and (N) represents the total number of samples taken along the ray.

Once each distance (p_i) for a 3D sample point along the ray, its corresponding 3D sample point coordinates in the GCS can be calculated using Equation (10):

$$\boldsymbol{x_i} = \boldsymbol{r_k}(\boldsymbol{i}) = \boldsymbol{r_k}(0) + p_i \cdot \boldsymbol{d_k}, \tag{10}$$

where $\boldsymbol{x}_i = [x_{G,i}, y_{G,i}, z_{G,i}]^T$ represents the 3D sample point's coordinate on the ray at a sampled position p_i, $\boldsymbol{d_k}$ is the ray's directional unit vector for the kth pixel, and $\boldsymbol{r_k}(0)$ is $\boldsymbol{t} = [t_x, t_y,\ t_z]$ in GCS because $\boldsymbol{t}$ is the camera position in GCS. Given that N, and each $\boldsymbol{x_i}$ comprises 3 elements, a total $3 \times N$ vector is inputted into the positional encoder as illustrated in. In this encoder, the input data undergoes a positional encoding process before being fed into the DNN-I and -II to enhance the model's ability to render scenes with high-frequency details about different directions and distances. This positional encoding expands the dimensions of each 3D sample point $\boldsymbol{x}_i = [x_{G,i}, y_{G,i}, z_{G,i}]^T$ and its associated ray direction $\boldsymbol{d_k}$ to $(N \times 3 \cdot 2E_{\boldsymbol{x}})$ and $(N \times 3 \cdot 2E_{\boldsymbol{d}})$ dimensions, respectively, where $E_{\boldsymbol{x}} = 10$ and $E_{\boldsymbol{d}} = 4$ are the positional encoding factors for $\boldsymbol{x}_i$ and $\boldsymbol{d_k}$, respectively. Therefore, the input dimensions are $3 \cdot (N) \times 2(E_{\boldsymbol{x}} = 10) = N \times 60$ and $N \times 3 \cdot 2(E_{\boldsymbol{d}} = 4) = N \times 24$, respectively, where "2" accounts for the sine and cosine cases in Equations (11) and (12). The conceptual basis for the expansions of $\boldsymbol{x}_i$ and $\boldsymbol{d_k}$ is founded on a collection of sine and cosine curves as adopted in the

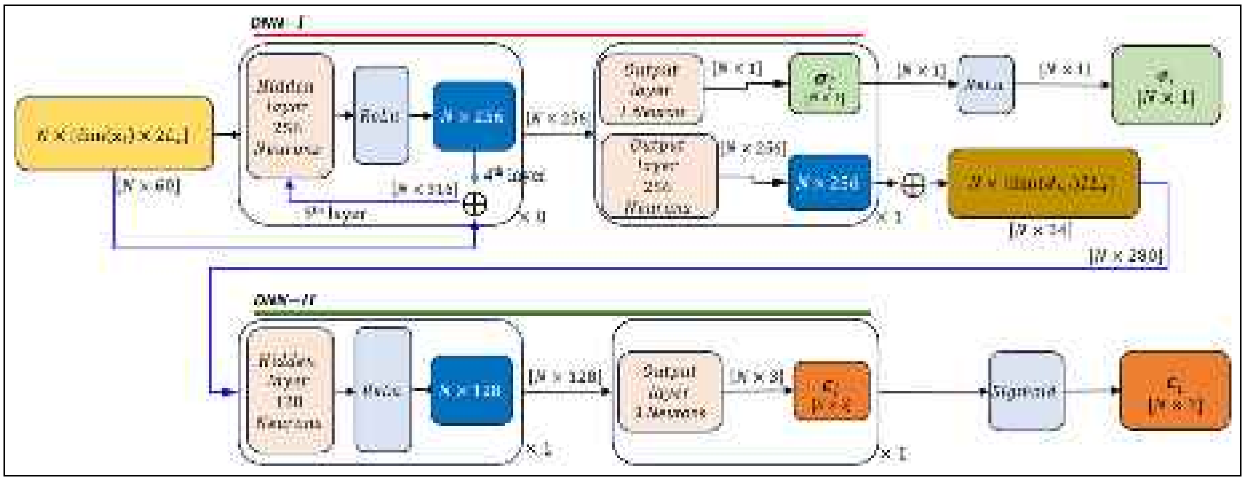


Figure 7. The architectures of DNN-I and -II in the NeRF Model.
DNN: deep neural network; NeRF: neural radiance fields.

Fourier series, and the conceptual expressions of the positional encoders are expressed in Equations (11) and (12):

$$\boldsymbol{\gamma}(\boldsymbol{x}_i, E_x) = [\sin(2^0\pi\boldsymbol{x}_i),\ \cos(2^0\pi\boldsymbol{x}_i),\ \cdots,\ \sin(2^{E_x-1}\pi\boldsymbol{x}_i),\ \cos(2^{E_x-1}\pi\boldsymbol{x}_i)] \tag{11}$$

$$\boldsymbol{\gamma}(\boldsymbol{d}_k, E_d) = [\sin(2^0\pi\boldsymbol{d}_k),\ \cos(2^0\pi\boldsymbol{d}_k),\ \cdots,\ \sin(2^{E_d-1}\pi\boldsymbol{d}_k),\ \cos(2^{E_d-1}\pi\boldsymbol{d}_k)] \tag{12}$$

Through this positional encoding, by enhancing the model's sensitivity to subtle variations in geometry and appearance, NeRF is capable of learning RGB values across a diverse range of distances and angles, enabling the production of photorealistic images.

DNN-I and DNN-II for prediction of RGB color and density. The expanded inputs ($N\times3{\cdot}2E_x$) and ($N\times3{\cdot}2E_d$) of the positional and directional vectors are processed by DNN-I and DNN-II to predict the color and density at each sampled 3D point along the ray, as shown in Figure 5. The output of each DNN includes a 4D vector containing RGB color ($\boldsymbol{c}_i$) and density ($\boldsymbol{\sigma}_i$). Density here indicates the predicted transparency or opaqueness of a point in 3D space. For example, a low density near a glass surface allows background objects to be visible, whereas a high density near a cement wall makes it opaque, obscuring objects behind it.

Figure 7 illustrates the architectures of DNN-I and DNN-II. These DNNs consist of fully connected layers with 256 and 128 neurons, respectively, followed by rectified linear unit (ReLU). Specifically, after the 4th layer, a skip connection reintroduces the input to the 5th layer, and the feature map ($N\times256$) produced in the 4th layer is concatenated with the original input ($N\times60$). By the output layer of DNN-I, the color density $\boldsymbol{\sigma}$ ($N\times1$) and the ($N\times256$) dimensional feature vector are produced simultaneously. A ReLU activation is applied to the density $\boldsymbol{\sigma}_i(N)$ to ensure non-negativity.

Afterward, the expanded input ($N\times3{\cdot}2E_d$) is concatenated with the ($N\times256$) dimensional feature vector to pass through an additional layer with 128 neurons and a ReLU activation function. This is then passed to the final output layer of DNN-II with a sigmoid activation function to predict the RGB color value, ($\boldsymbol{c}_i$), at the ith sample point coordinate ($\boldsymbol{x}_i = [x_{G,i}, y_{G,i}, z_{G,i}]^T$). The resulting 4D vector ($\boldsymbol{c}_i$, $\boldsymbol{\sigma}_i$) represents the predicted RGB color and density value of the ith sampled point along the ray.

Volume rendering and loss function. In the NeRF model, each 3D point's color and density along a ray contribute to the final image color, processed through a volumetric rendering function as shown in Figure 5. This volume rendering function calculates the RGB color at the kth pixel as a weighted sum of the colors from points along the ray, as shown in Equation (13).

$$\hat{\boldsymbol{C}}(\boldsymbol{k}) = \sum_{i=1}^{N} \boldsymbol{w}_i\boldsymbol{c}_i, \tag{13}$$

$$\text{where } w_i = \exp\left(-\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j\right)(1-\exp(-\boldsymbol{\sigma}_i\delta_i)) \tag{14}$$

where $\hat{\boldsymbol{C}}(\boldsymbol{k})$ is the estimated color of the kth pixel, N is the number of sampled 3D points $\boldsymbol{x}_i$ along the ray, $\boldsymbol{c}_i$ is the color of the ith sampled 3D point, w_i represents the weight assigned to the color $\boldsymbol{c}_i$, and, $\boldsymbol{\sigma}_i$ denotes the density at the ith sampled 3D points $\boldsymbol{x}_i$, and δ_i is the distance between the ith and $i-1$th sample points.

The first term $\exp\left(-\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j\right)$ in Equation (14) calculates the accumulated transparency from the first sample point to just before the ith sampled point, signifying how previous densities along the ray obscure or allow the current 3D point's color to contribute to the pixel's overall color. If up to the $i-1$th densities are high, $\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j$ will be high, which will lead $\exp\left(-\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j\right)$ to be low, lowering the entire weight of the ith point. Conversely, if up to the $i-1$th densities are low, then the ith point might have high density. Therefore, $\exp\left(-\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j\right)$ will be high. Again, a high sum of densities at preceding points reduces the importance of the current point, as it implies that the current point is less visible compared to those in front of it. Conversely, a low density at preceding points increases the significance of the current point, as it suggests transparency in the preceding points.

The second term $(1-\exp(-\boldsymbol{\sigma}_i\delta_i))$ represents the probability of light interacting with the point either by absorption or scattering at the ith 3D point. If the density $\boldsymbol{\sigma}_i$ at the ith point is high, $\exp(-\boldsymbol{\sigma}_i\delta_i)$ will be small, and $(1-\exp(-\boldsymbol{\sigma}_i\delta_i))$ will be large. This ensures that points with high density contribute more to the final color, as they have higher opacity and thus scatter or emit more light.

Overall, combining these terms, the weight w_i is a product of the accumulated transmittance and the probability of interaction at the current point. It also expresses the distance between the ith 3D point and the first sample point. If the point is close to the ray starting point, then the accumulation $\sum_{j=1}^{i-1}\sigma_j\delta_j$ will be relatively low, which will lead $\exp\left(-\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j\right)$ to be high. Conversely, if the point is far, $\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j$ will be relatively high, then $\exp\left(-\sum_{j=1}^{i-1}\boldsymbol{\sigma}_j\delta_j\right)$ will become low, which will cause the weight w_i to decrease. This ensures that points closer to the viewer with higher densities contribute more to the final color. Points further away or in regions with lower densities contribute less due to attenuation.

PDF sampling and loss function. As shown in Figure 5, for density value prediction, two separate sampling methods, coarse and fine, are implemented. The different density values obtained are then used to calculate separate predicted $(\hat{\boldsymbol{C}}(\boldsymbol{k}))$ values, labeled as $(\hat{\boldsymbol{C}}_c(k))$ for coarse predictions and $(\hat{\boldsymbol{C}}_f(k))$ for fine predictions. Consequently, for a coarse estimation of the density value, $(\boldsymbol{\sigma}_i)$, $N = 64$ sample points are selected through uniform random sampling according to Equation (9) to predict the density value for $\hat{\boldsymbol{C}}_c(k)$. The process then advances to employ PDF sampling, a strategy designed to select significant points of higher relevance, to predict the density value for $(\hat{\boldsymbol{C}}_f(k))$. This targeted method facilitates a later training phase focused on these crucial points, thereby enhancing the model's efficiency.

The PDF sampling process incorporates a PDF for normalization, as outlined in Equation (15):

$$\mathrm{PDF}_i = \hat{w}_i = \frac{w_i}{\sum^{w}} \tag{15}$$

Subsequently, the cumulative distribution function (CDF) is used to calculate cumulative values, as shown in Equation (16):

$$\mathrm{CDF}_i = \sum_{j=1}^{i} \hat{w}_j \tag{16}$$

Following this, 128 random values are generated, $(u_\tau \sim \mho(0,1) = \{u_1, u_2 \cdots u_{128}\})$, where q ranges from 1 to 128. For fine-tuning, 128 new sample points are selected, each corresponding to a specific interval determined by the CDF. For example, a random value (u_τ) falls within an interval $(\mathrm{CD}F_{N=1} \leqslant u_q \leqslant \mathrm{CD}F_{N=N})$, after which the points (p_i) and (p_{i+1}) are identified. Within this interval, a new sampling distance (p'_τ) is calculated using interpolation, as per Equation (17):

$$p'_\tau = p_i + \left(\frac{u_\tau - \mathrm{CDF}_i}{\mathrm{CDF}_{i+1} - \mathrm{CDF}_i}\right)\cdot(p_{i+1} - p_i) \tag{17}$$

This PDF sampling method generates new distances $\{p'_1, p'_2 \cdots p'_{128}\}$. Based on this, new 3D sample points are calculated using the same equation as before. These points undergo the process similar to the initial coarse prediction, leading to new color and density values for the 128 sample points. Using the volume rendering formula (Equation (13)), the $(\hat{C}_f(k))$ value is predicted for a fine-tuned estimation.

Finally, using the two predicted values $(\hat{\boldsymbol{C}}_c(k))$ and $(\hat{\boldsymbol{C}}_f(k))$, the loss is calculated against the actual pixel colors $(\boldsymbol{C}(k))$ from the dataset using the mean squared error (MSE), detailed in Equation (18):

$$\mathcal{L} = \sum_{k\in\mathbb{K}} \left[\|\hat{\boldsymbol{C}}_c(k) - \boldsymbol{C}(k)\|_2^2 + \|\hat{\boldsymbol{C}}_f(k) - \boldsymbol{C}(k)\|_2^2 \right] \tag{18}$$

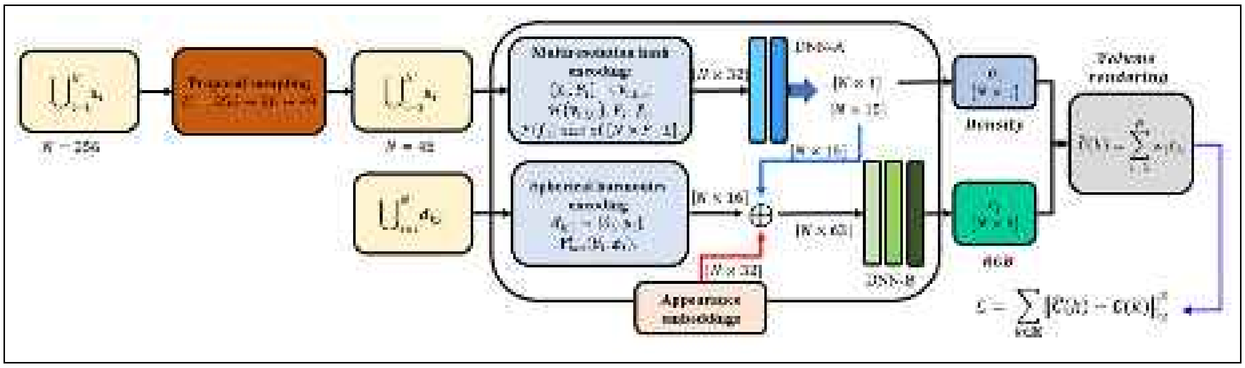


Figure 8. Nerfacto model architecture.

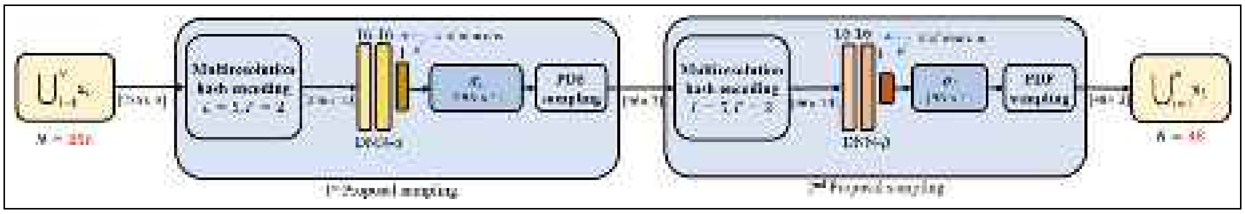


Figure 9. Proposal sampling network structure.

where $\mathbb{K}$ represents the set of all pixels.

From this methodology, it's understood that having a larger number of similar images of an object increases the likelihood of consistently observing the same color. This consistency in color observation is vital as it leads to a higher weighting in the training process, thereby improving the accuracy of the model's actual color representation.

Nerfacto model. As we discussed above, the original NeRF model faces several limitations: (1) Computational efficiency: NeRF requires significant computational resources due to the large number of parameters in the DNN and the high number of samples per ray, making it slow to train and render, (2) Memory usage: The large DNNs and the need to store many samples per ray leads to high memory consumption, (3) Training speed: The original NeRF is slow to train due to its large DNN and the high number of samples per ray, and (4) Flexibility and adaptability: NeRF model's original architecture limits its adaptability to different types of scenes and datasets.

To overcome these limitations, the Nerfacto model adopts proposal sampling from Mip-NeRF 360 to efficiently sample 3D points, multiresolution hash encoding from Instant-NGP to replace the positional encoding of $\boldsymbol{x}_i$, and spherical harmonics encoding from Ref-NeRF to replace the positional encoding of $\boldsymbol{d}_k$. Additionally appearance embedding technique from NeRF-W is added to DNN-B as shown in Figure 8. The detail of each module is described in subsections.

Proposal sampling. The traditional NeRF sampling process involves uniformly sample points along each ray having $N = 64$, processing them through a DNN, conducting PDF sampling, and then processing through the same DNN again, as detailed in section "Neural radiance fields." This method can be computationally intensive and inefficient, especially for large and complex scenes. To address these drawbacks, Nerfacto introduces proposal sampling, as depicted in Figure 8. This approach allows for the preselection of important 3D sample points ($N = 48$) from initial ($N = 256$) points before they are processed by DNN-A, as shown in Figure 8.

Figure 9 provides a detailed view of the proposal sampling module's architecture. This module comprises two distinct DNNs (namely, DNN-α and DNN-β), each consisting of three layers. Initially, the input ($\boldsymbol{x}_i$) undergoes a multiresolution hash encoding process, which will be elaborated on in the following section. The output from this hash encoding is then input into DNN-α, which predicts ($\boldsymbol{\sigma}_i$), followed by a PDF sampling process that yields 96 new 3D sample points. This proposal sampling process is repeated with DNN-β as the second proposal sampling stage, resulting in an

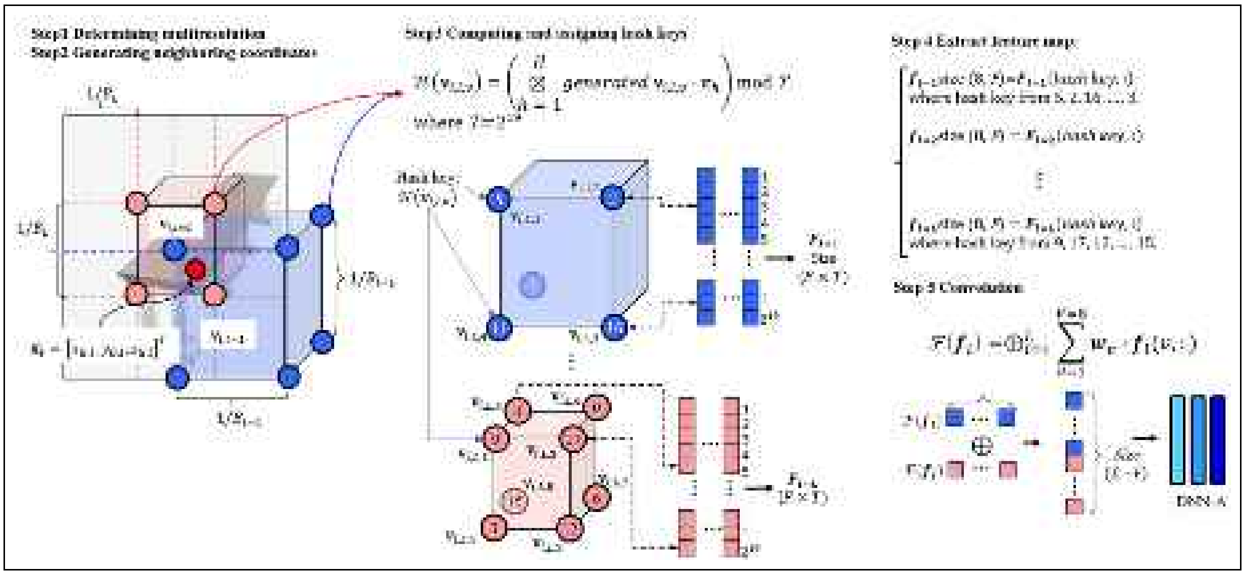

Step1 Determining multiresolution
Step2 Generating neighboring coordinates
Step3 Computing and assigning hash keys
where $T=2^{19}$
Step4 Extract feature map
Step5 Concatenation
DNN-A

Figure 10. Schematic view of the multiresolution hash encoding process.

additional 48 sample points, as illustrated in Figure 9. Consequently, the final count of 3D sample points for Nerfacto is ($N = 48$).

By leveraging this method, the proposal sampling approach aids in identifying high-density samples through DNNs and PDF, thereby effectively decreasing the total number of 3D sample points to $N = 48$. This reduction plays a crucial role in diminishing the computational expenses for the entire Nerfacto model. This is particularly notable when contrasted with the NeRF model that employs $N = 128$ sample points.

Multiresolution hash encoding. In traditional NeRF, pipelines as described in the previous section, the input dimensions of the positional encoders are determined by $N \times 3 \cdot 2E_x$ that is the number, N, of sample points in a ray and its number, 3, of coordinates, and these are augmented by $2E_x$ to learn broad ranges of distances and angles. However, the size of the augmented input is too large [i.e., $N \times 3 \cdot 2(E_x = 10) = N \times 60$]; therefore, Nerfacto adopts a multiresolution hash encoding method. Multiresolution hash encoding method generates the input of the DNN-A based on multi-level (resolution) voxel sizes that include the specific sample point, $\boldsymbol{x_i}$, in a ray as shown in Figure 10.

As shown in Figure 10, as the first step, in a 3D space of $\boldsymbol{x_i}$ that is a sample point in a ray of a specific *kth* pixel, different levels (sizes) of voxels are generated using the grid size ratio B_l at each level l where it has a maximum level L, which determines the resolution (size), is calculated using the Equation (19).

$$B_l := B_{\min} \cdot b^l \ , \tag{19}$$

where $\lfloor \cdot \rfloor$ denotes the floor function, $:=$ means that "it defines", B_{min} is the base resolution, and b is a growth factor, which can be calculated by Equation (20):

$$b := \exp\left(\frac{\ln B_{\max} - \ln B_{\min}}{L - 1}\right), \tag{20}$$

where $B_{\min}$, and $B_{\max}$, are defined by the user, $B_{\min}$ defines the greatest voxel size, and $B_{\min}$ defines the smallest voxel size. As examples, $B_{\min}$ and $B_{\max}$ are commonly defined as 16 and 2048 in Nerfacto, respectively. These voxels are generated in a 3D pseudo space at the global X–Y plane of $\boldsymbol{x_i}$ as shown in Figure 10. The maximum level, L, also can be randomly defined, and in this study, it is 16.

For a given 3D sample point having $\boldsymbol{x_i} = [x_{G,i}, y_{G,i}, z_{G,i}]^T$, 3D coordinates for eight neighboring (voxel) vertices are generated by the voxel size at each grid level (resolution) as shown in Figure 10. Each vertex coordinate will be calculated by the Equation (21).

$$\lfloor \boldsymbol{x}_{i,l} \rfloor := \lfloor \boldsymbol{x}_i \cdot B_l \ \rfloor,$$

$$\lfloor \boldsymbol{x}_{i,l} \rfloor := \lfloor \boldsymbol{x}_i \cdot B_l \rfloor$$

$$\text{Resulted in}: \ \boldsymbol{v}_{i,l,v} = [\boldsymbol{v}_{i,l,1} = (\lfloor x_{G,i,l} \rfloor, \lfloor \ y_{G,i,l} \rfloor, \lfloor \ z_{G,i,l} \rfloor),$$
$$\boldsymbol{v}_{i,l,2} = (\lfloor x_{G,i,l} \ \rfloor, \lfloor \ y_{G,i,l} \rfloor, \ \lfloor z_{G,i,l} \rfloor),$$

$$\begin{aligned}\boldsymbol{v}_{i,l,3} &= (\lfloor x_{G,i,l}\rfloor,\ \lfloor y_{G,i,l}\rfloor,\ \lfloor z_{G,i,l}\rfloor),\\ \boldsymbol{v}_{i,l,4} &= (\lfloor x_{G,i,l}\rfloor,\ \lfloor y_{G,i,l}\rfloor,\ \lfloor z_{G,i,l}\rfloor),\\ \boldsymbol{v}_{i,l,5} &= (\lfloor x_{G,i,l}\rfloor,\ \lfloor y_{G,i,l}\rfloor,\ \lfloor z_{G,i,l}\rfloor),\\ \boldsymbol{v}_{i,l,6} &= (\lfloor x_{G,i,l}\rfloor,\ \lfloor y_{G,i,l}\rfloor,\ \lfloor z_{G,i,l}\rfloor),\\ \boldsymbol{v}_{i,l,7} &= (\lfloor x_{G,i,l}\rfloor,\ \lfloor y_{G,i,l}\rfloor,\ \lfloor z_{G,i,l}\rfloor),\\ \boldsymbol{v}_{i,l,8} &= (\lfloor x_{G,i,l}\rfloor,\ \lfloor y_{G,i,l}\rfloor,\ \lfloor z_{G,i,l}\rfloor)],\end{aligned} \quad (21)$$

where $\lfloor\cdot\rfloor$ and $\lfloor\cdot\rfloor$ denote the floor and ceiling functions, respectively, applied to the scaled coordinates to derive the integer indices of the voxel vertices, and $v(=8)$ is the number of vertices of a voxel. Then, Equation (21) generates eight combinations of voxel corner coordinates that are only the surrounding the specific sample point.

As shown in Figure 10, these eight generated coordinates are then converted into hash key values $\mathcal{H}(\boldsymbol{v}_{i,l,v})$ for hash table $\boldsymbol{F}_l$ using hash function $\mathcal{H}$ as presented in Equation (22):

$$\mathcal{H}(\boldsymbol{v}_{i,l,v}) = \left(\underset{h=1}{\overset{H}{\boxtimes}} \text{ generated } \boldsymbol{v}_{i,l,v}\cdot\pi_h\right) \bmod T, \quad (22)$$

where l is the resolution level, $H(=3)$ is the number of 3D coordinate dimension, π are large prime numbers, $\boxtimes$ denotes bitwise XOR operation. The values of a hash table, $\boldsymbol{F}_l$, having a size of $F\times T$ are randomly initialized as shown in Figure 10. Here $F=2$, and $T=2^{19}$, but it can be any value if T is at least eight to host eight hash key's features. The size of each hash key's feature map is also randomly defined by the user. In this study, $\pi_1 = 1$, $\pi_2 = 2,654,435,761$, and $\pi_3 = 805,459,861$. These π values and very large T are selected to avoid hash key collision, which prevents overlapping or getting the same hash key for each different vertex for a voxel that is in a specific resolution. Therefore, each vertex has different hash key, and different hash key has different feature values, $\boldsymbol{f}_l$, that are coming from the randomly generated hash table values as shown in Figure 10.

To compute the final feature vectors, $\mathcal{F}(\boldsymbol{f}_l)$, convolution is performed on the feature vectors located at each corner of the voxel defined by the position of $\boldsymbol{x}_i$ in its space. This involves using the calculated feature vectors, $\boldsymbol{f}_l$, and their respective weights, $\boldsymbol{w}_v$, to determine the final interpolated feature vector, which serves as input for the DNN. The formula for convolution, including concatenation, $\oplus$, is described in Equation (23).

$$\mathcal{F}(\boldsymbol{f}_l) = \oplus_{l=1}^{L}\sum_{v=1}^{V=8}\boldsymbol{w}_v\cdot\boldsymbol{f}_l(v,\ :)$$

$$\begin{aligned}&\text{where } \boldsymbol{w}_0 = (1-w_x)(1-w_y)(1-w_z),\\ &\boldsymbol{w}_1 = w_x(1-w_y)(1-w_z),\\ &\boldsymbol{w}_2 = (1-w_x)w_y(1-w_z),\ \boldsymbol{w}_3 = (1-w_x)(1-w_y)w_z,\\ &\boldsymbol{w}_4 = w_xw_y(1-w_z)\ \boldsymbol{w}_5 = w_x(1-w_y)w_z,\\ &\boldsymbol{w}_6 = (1-w_x)w_yw_z,\ \boldsymbol{w}_7 = w_xw_yw_z,\end{aligned} \quad (23)$$

where $\boldsymbol{w}$ convolution weight for each dimension, is calculated using Equation (24).

$$\begin{aligned}\boldsymbol{w_l} &:= \boldsymbol{x}_{i,l} - \boldsymbol{x}_{i,l} = \boldsymbol{x}_i\cdot B_l - \boldsymbol{x}_i\cdot B_l\\ \boldsymbol{w_l} &= (w_x,\ w_y,\ w_z)\end{aligned} \quad (24)$$

where $\boldsymbol{w}$ includes the coordinates of the sample point $\boldsymbol{x}_{i,l}$, therefore, the generated input with a size of $L{\cdot}F$ are dependent on the 3D coordinate of the specific sample point in a ray. Since Nerfacto uses $N=48$, the eventually generated input size of the DNN-I is $N\times L{\cdot}F$. This process is independently executed at each of the L levels.

The use of hash encoding in the Nerfacto model significantly enhances computational efficiency by reducing the granularity of data that the DNN processes, without compromising the integrity of spatial information. This method enables the handling of large-scale environments and complex geometries more effectively due to its multi-resolution approach and efficient memory usage. However, while hash encoding effectively manages spatial data, it does not inherently account for the complex interactions of light and color in a scene in terms of different viewing directions, which are crucial for achieving photo-realistic renderings. This leads the Nerfacto model to incorporate spherical harmonics encoding.

Spherical harmonics encoding. The Nerfacto model uses spherical harmonic functions to generate the input for the DNN-II using the viewing direction $\boldsymbol{d_{k,i}} = [d_x,\ d_y, d_z]^T$. From the $\boldsymbol{d_{k,i}}$, new viewing directional angles, θ and ϕ, in X–Y plane and Z plane, which are spherical coordinate system (SCS), are determined, respectively, by Equation (25).

$$\begin{aligned}\theta_i &= \tan^{-1}\left(\frac{d_y}{d_x}\right)\\ \phi_i &= \cos^{-1}\left(\frac{d_z}{d_x^2+d_y^2+d_z^2}\right)\end{aligned} \quad (25)$$

The value, $\boldsymbol{Y}_{sm}^{\boldsymbol{i}}(\theta_i,\ \phi_i)$, of spherical harmonics for a given angle is computed using Equation (26), which

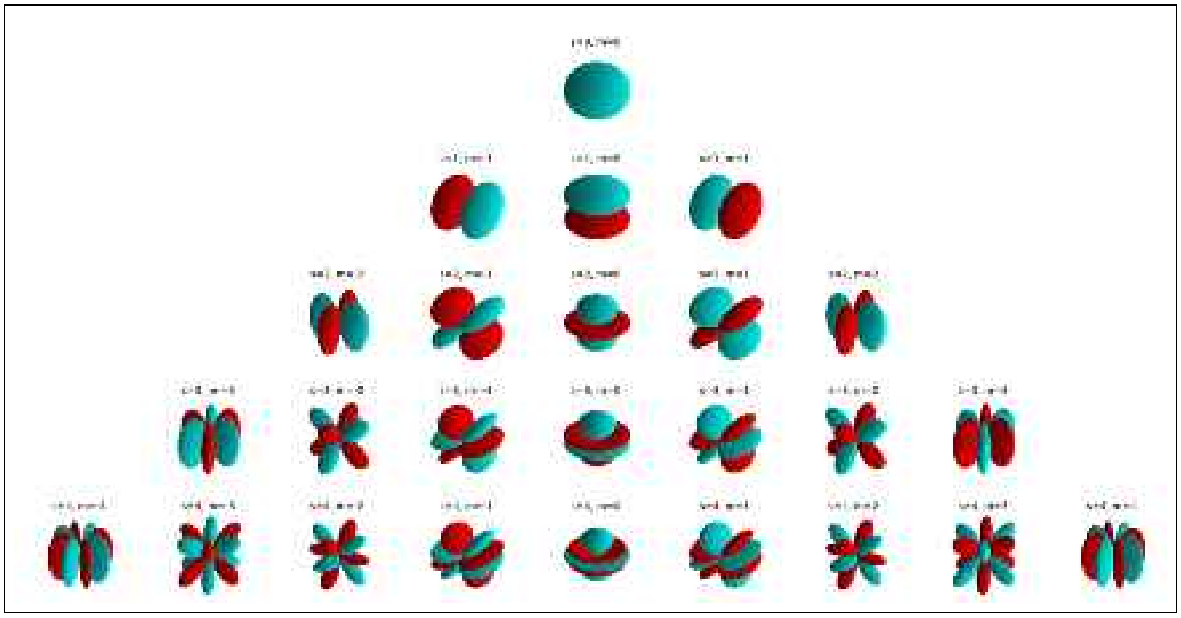


Figure 11. Visualization of spherical harmonic functions.

enables encoding of complex directional information based on the angular variations of the ray direction.

$$\boldsymbol{Y}^{\boldsymbol{i}}_{\boldsymbol{sm}}(\theta_i,\ \phi_i) = \begin{cases} (-1)^m\sqrt{\frac{2s+1}{4\pi}\frac{(s-m)!}{(s+m)!}}\,P_s^m(\cos\phi_i)e^{im\theta_i}, & \text{if } m\geqslant 0 \\ (-1)^m\sqrt{\frac{2s+1}{4\pi}\frac{(s-m)!}{(s+m)!}}\,P_s^{-m}(\cos\phi_i)e^{-im\theta_i}, & \text{if } m<0 \end{cases}, \tag{26}$$

where s is the degree of spherical harmonic function (in Nerfacto, $s = 3$), m denotes an integer from $-s$ to s, and P_s^m is the associated Legendre polynomials, defined as in Equation (27).

$$P_s^m\ (x) = \left(1-x^2\right)^{\frac{m}{2}}\frac{d^m}{dx^m}\left(\frac{1}{2^s s!}\frac{d^s}{dx^s}\left(x^2-1\right)^s\right) \tag{27}$$

For a given degree s, there are $2s + 1$ possible values of m, resulting in $2s + 1$ spherical harmonic functions. To determine the total number of spherical harmonic functions up to a maximum degree s, the numbers of functions are summed up for each degree from 0 to s: $1 + 3 + 5 + \ldots + (2s + 1)$ as shown in Figure 11.

For example, ($s = 0$, $m = 0$), ($s = 1$, $m = -1, 0, 1$), ($s = 2$, $m = -2, -1, 0, 1, 2$), ($s = 3$, $m = -3, -2, -1, 0, 1, 2, 3$), which can be expressed as $\sum_{k=0}^{s}(2k+1) = (s+1)^2$. In this article, $s = 3$, and therefore for each sample point, the total number of spherical harmonic functions made is 16-dimensional representation of the ray direction, capturing complex angular dependencies. Therefore, the spherical harmonic encoder is broadcasted to an input size of $N(N = 48$ for Nerfacto$)\times 16$.

To sum up, the original NeRF model employs positional encoding to extend the dimensionality of input coordinates. This encoding uses a series of sinusoidal functions to map low-dimensional coordinates into a higher-dimensional space, capturing high-frequency variations and enabling the model to represent complex patterns and details in the scene. For example, for the original NeRF model with $E_d = 4$ frequency bands, the output dimension for a ray direction becomes $N\times\dim(\boldsymbol{d}_{k,i})\cdot 2E_{\boldsymbol{d}} = N(N = 64$ for original NeRF$)\times 24$. In contrast, Nerfacto uses spherical harmonic encoding to encode the direction of rays. Spherical harmonics provide a compact and efficient representation of angular information, making them ideal for capturing view-dependent effects and complex lighting variations. By converting 3D direction vectors, $\boldsymbol{d}_{k,i}$, into spherical coordinates, $[\theta_i,\ \phi_i]$, and applying spherical harmonic functions, $\boldsymbol{Y}^{\boldsymbol{i}}_{\boldsymbol{sm}}(\theta_i,\ \phi_i)$, Nerfacto encodes directional data in a 16-dimensional space when using $s = 3$ spherical harmonics and resulted in the input size of $N(N = 48$ for Nerfacto$)\times 16$ which is much smaller than that of the original NeRF.

Appearance embedding. The original NeRF model trains its second DNN (DNN-B) using each pixel's information independently during the backpropagation of the network, without considering the entire image

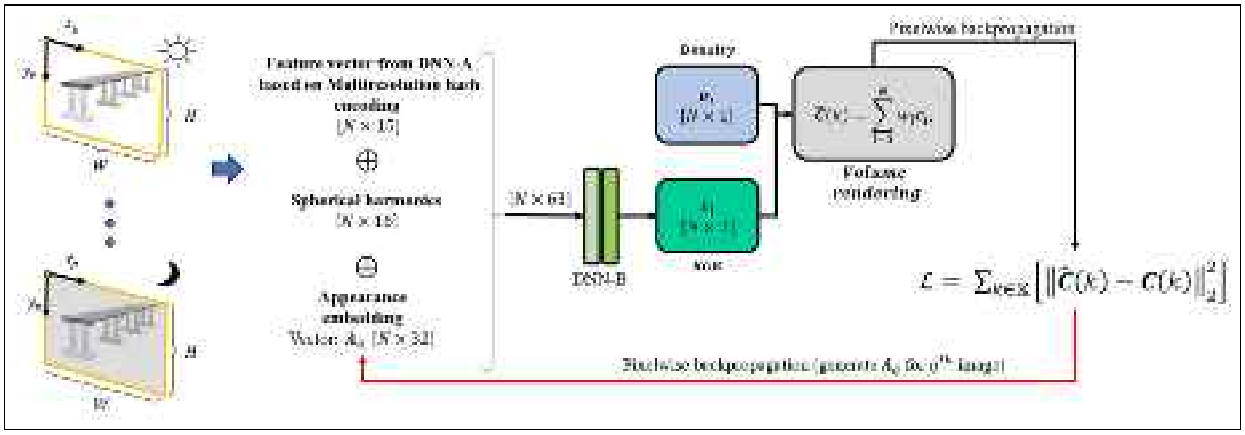

Feature vector from DNN-A based on Multiresolution hash encoding [N × 15]
Spherical harmonics [N × 16]
Appearance embedding Vector: A_q [N × 32]
[N × 63]
DNN-B
Density
Volume rendering
Pixelwise backpropagation
Pixelwise backpropagation (generate A_q for q^{th} image)
$\mathcal{L} = \sum_{k \in \mathcal{K}} \left[\|\hat{C}(k) - C(k)\|_2^2 \right]$

Figure 12. Visualization of the appearance embedding process.

context because it is a pixelwise comparison in the loss function as presented in Equation (15). This approach makes it image-independent, relying solely on the ray direction and position for color prediction. This structure may cause the model to misinterpret features when there are images of the same object taken under different lighting conditions, potentially resulting in artifacts or inconsistencies in the 3D reconstruction.

In contrast, the Nerfacto model incorporates an additional input for the DNN-B that is the appearance embedding. This additional technique enables Nerfacto to handle varying conditions within the training dataset, such as different lighting setups, as shown in Figure 12. The appearance embedding generates a feature vector $\boldsymbol{A}_q$ for each image q, which is initialized with a vector with size of $[N \times 32]$ of learnable parameters (in Nerfacto). It is a fixed-size vector for each image.

It is concatenated with the outputs from the spherical harmonics of the ray direction ($N \times 16$) and the feature vector ($N \times 15$) from the DNN-A based on the multiresolution hash encoding. The final concatenated input size to be fed into DNN-B is $N(N = 48) \times 63$. Following this, the DNN-B processes this vector, and the loss function (Equation (15) based on the color of each pixel, is used for updates. The learnable parameters of the $\boldsymbol{A}_q$ is also pixelwise updated, but for each image, different A is generated and updated. This process repeats correspondingly for an image size to capture detailed appearance features, such as lighting variations. For inference, it takes the average of all feature vector A. This method renders the Nerfacto model image-dependent, enabling it to discern differences between images, which enhances its adaptability in managing diverse lighting environments. As a result, the model can produce a more consistent and coherent 3D reconstruction, where the lighting and colors are better matched across different parts of the object.

Table 1. Datasets for crack segmentation case studies.

Cases	Image size (px)	Number of images
STRNet (Training)	1024 × 512	1203
STRNet (Test)	1024 × 512, 1280 × 720	545
STRNet + TTA (Test)	2992 × 2000	11

STRNet: semantic transformer representation network; TTA: test time augmentation.

Data

The foundation of any successful structural damage identification and 3D reconstruction project lies in the quality of the data acquired. This section discusses the data used in STRNet with TTA for crack segmentation and the data used in 3D image reconstruction for 3D damage mapping, detailed in sections "Data for crack segmentation" and "3D image reconstruction including damage mapping," respectively.

Data for crack segmentation

For damage segmentation, the pretrained STRNet was utilized. STRNet was trained using diverse datasets with varying resolutions (1024 × 512 and 1280 × 720), initially acquired by Kang and Cha,[21] the developers of STRNet. Table 1 provides details on the number of images and their sizes for each case.

3D image reconstruction including damage mapping

To generate a 3D image reconstructed model, high-quality RGB images of the structure of interest are required. As discussed in section "Camera parameter

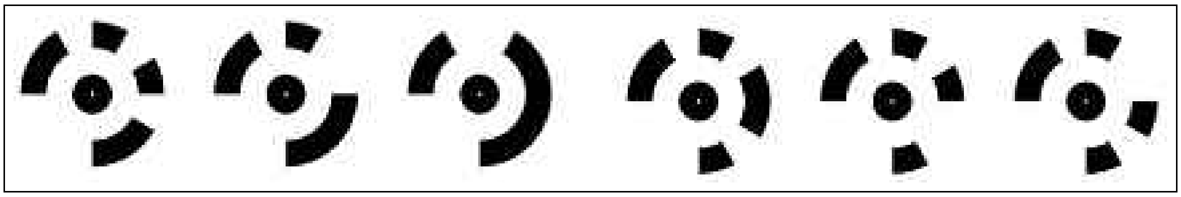


Figure 13. Markers.

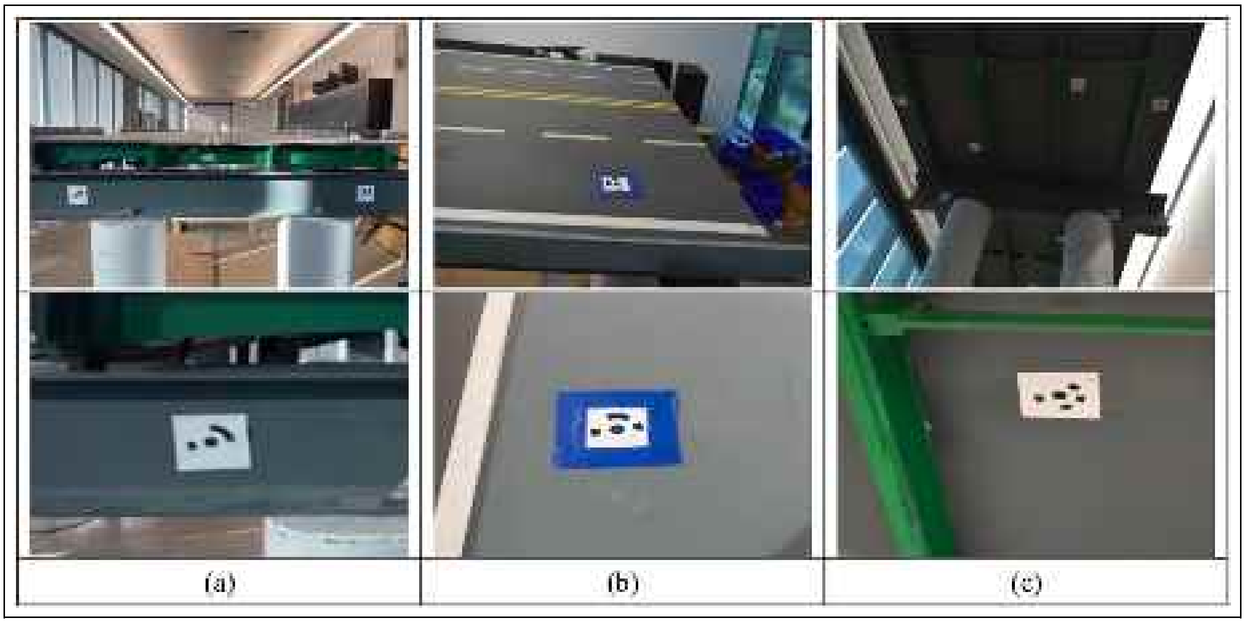


Figure 14. Aligned Markers on the bridge. (a) Front view of the bridge. (b) Top view of the bridge and (c) Bottom view of the bridge.

extraction," each image has its own local coordinate system (LCS) that must be converted to the global coordinate system (GCS) for the 3D reconstructed model in the Nerfacto model. Intrinsic and extrinsic parameters ($\boldsymbol{K}$ and $\boldsymbol{R}|\boldsymbol{t}$) should be extracted for this conversion. Metashape was implemented to facilitate this process, using unique markers for more accurate extraction of these parameters, as shown in Figure 13.

These markers, created with Metashape software, were carefully positioned on the top, bottom, and sides of the bridge, as depicted in Figure 14. They serve a dual purpose: providing precise reference points for the coordinate system and scale definition and facilitating effective camera alignment by providing a unique feature of each marker in each image. The markers, which are coded targets, were placed in the scene prior to taking photos. They were then utilized within the photo scan software, not only as reference points but also as crucial elements in the image matching process, particularly through the "Align Selected Cameras" feature. Adhering to best practices, we ensured that each captured image included at least one marker. This method greatly enhanced feature matching and spatial orientation, a critical aspect in photogrammetry. The use of markers proved especially advantageous for the Nerfacto technique, as it provided highly accurate camera pose estimations, crucial for precise 3D reconstruction.

To obtain high-resolution images, we used a Nikon digital camera. Table 2 details the camera settings used for data collection and the total number of images collected as input for Nerfacto for 3D reconstruction. To balance image quality with computational efficiency, we selected a resolution of 2992 $\times$ 2000 pixels and a focal length of 18 mm. Additionally, we chose the JPEG fine image format to balance image quality and file size, ensuring manageable data for subsequent processing stages. These settings ensured each image size was less than 5 MB, capturing high-quality images while considering computational constraints.

The process of capturing these images for 3D reconstruction involves meticulous attention to detail. Each

Table 2. Camera settings and collected RGB images.

Model name	Image size	# of images	Focal length	File format
Nikon D7200	2992 × 2000	1931	18 mm	JPEG

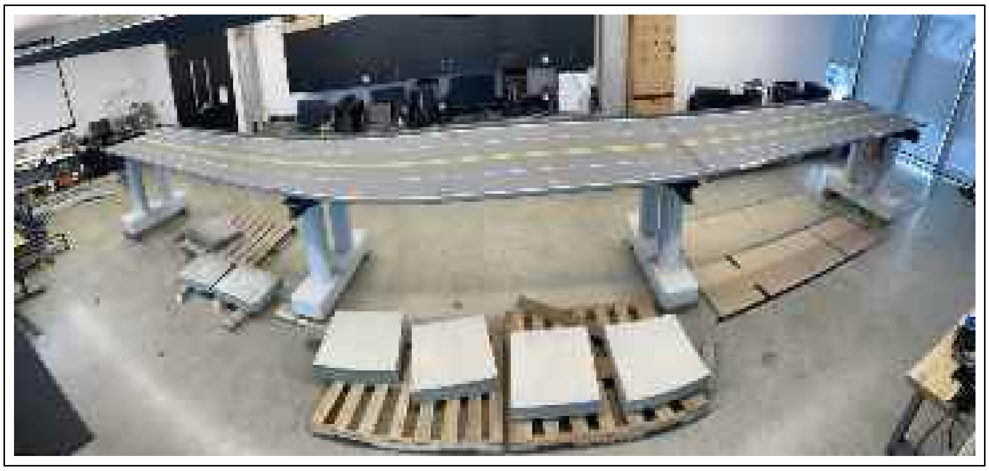


Figure 15. Panoramic view of the structure of interest.

photo was taken to capture the desired features of the bridge model, with close-ups ensuring that many image pixels represent the bridge. The experiment was conducted in the lab, as shown in Figure 15. To obtain an accurate 3D model, each image contained at least one marker, and significant emphasis was placed on image overlap, ensuring that each image overlapped with its neighboring images by at least 80%.

Case studies

In this section, we conduct extensive case studies to evaluate the performance of crack segmentation and subsequent 3D damage mapping on the 3D reconstructed model. These case studies aim to provide a comprehensive understanding of the various factors that affect the quality and accuracy of crack segmentation and 3D reconstruction for damage mapping.

Section "Crack generation and computing power" briefly describes the method used to induce cracks in real concrete bridge piers and outlines the computing power required for STRNet with TTA and Nerfacto simulations. Section "STRNet + TTA evaluation" details the various training and testing modes of STRNet with TTA techniques to achieve state-of-the-art results in crack segmentation, measured in terms of mean Intersection over Union (mIoU). Section "3D reconstruction and damage mapping using Nerfacto" presents various case studies for training the Nerfacto model and their results, including traditional methods of 3D reconstruction. Extensive comparative studies were conducted to demonstrate the effectiveness of our approach.

Through these case studies, we illustrate a real-world application of our integrated SHM approach, which combines advanced deep learning techniques for damage detection and 3D reconstruction for 3D damage mapping to create a digital twin with segmented damage information.

Crack generation and computing power

The bridge system used in this study has the following dimensions: a bridge deck width of 118 cm, a length of 921 cm with three simply supported deck systems, and a height of 124 cm for the bridge piers, including the thickness of the bridge deck. Cracks were induced in the bridge piers using a pressure machine to simulate damage, as illustrated in Figure 16. To conduct damage segmentation and 3D mapping on the 3D reconstructed model, we used a high-performance computer equipped with an AMD Ryzen 3990x CPU and an Nvidia RTX A6000 GPU, running on Ubuntu 20.04.6 LTS OS. For the Nerfacto model, we employed the Nerfstudio framework.

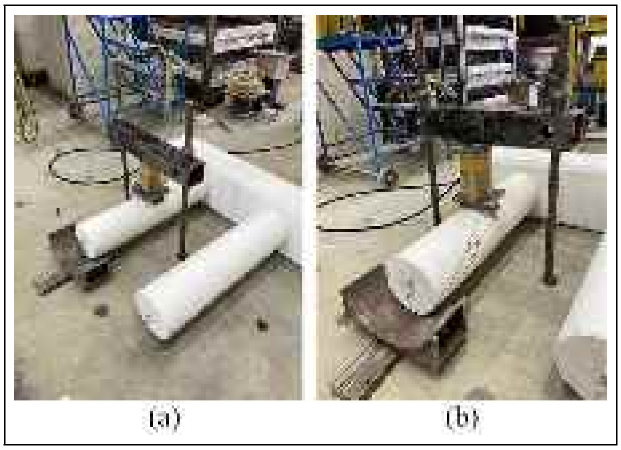


Figure 16. The process of creating cracks on the bridge pier. (a) Before creating cracks and (b) after creating cracks.

STRNet + TTA evaluation

In this article, the main focus is on 3D damage mapping through the 3D reconstruction of the bridge system. Therefore, the state-of-the-art crack segmentation method, STRNet, was employed for crack segmentation. mIoU was used as the evaluation metric to evenly consider false positives and false negatives, as presented in Equation (28).

$$\text{mIoU} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i + \text{FN}_i}\right) \tag{28}$$

where (N) is the number of images, (TP) represents True Positives, (FP) represents False Positives, and (FN) represents False Negatives.

The STRNet was trained using the existing dataset presented in Table 1 and achieved a 92% mIoU with a dataset of 545 images at resolutions of 1024 × 512 and 1280 × 720. However, as presented in Table 3, the crack segmentation accuracy of STRNet with this new structure is slightly lower than the reported performance in the original article,[21] at 80% mIoU. This reduction in accuracy is due to the significantly different structure and its environment compared to the existing data, which were collected from outdoor structures. Therefore, STRNet was integrated with the TTA approach to improve the results.

For the TTA, 12 different augmentation methods were conducted, including resizing, vertical and horizontal flipping, and rotations of 90°, 180°, and 270°. Through these augmentations, a total of 132 testing images were generated and tested. STRNet with TTA increased the mIoU to 90.3% with the new testing set of 132 images, excluding the 545 existing testing images from the original paper.[21] Cracks on the pillar were specifically selected to evaluate the mIoU for both STRNet and STRNet with TTA for this structure of interest.

Figure 17 visually demonstrates the performance of STRNet and the effects of TTA on crack segmentation accuracy through real cracks on the structure of interest. The figure shows the results of segmented cracks in the bridge piers and compares the traditional STRNet results with those of STRNet with TTA. As indicated by the blue-lined circles in the STRNet results, there are significant improvements in crack segmentation overall when using STRNet with TTA. These improvements stem from the nature of TTA.

TTA augments the input image in various ways, as described in section "Test time augmentation." The augmented input images are then fed into the trained STRNet, and all the results from the augmented input images are unified together. The unified results are presented as the final output. In this way, very thin cracks and missed cracks, which might be overlooked for various reasons, have a chance to be detected at least once, contributing to the final results. Consequently, STRNet with TTA achieved a 90.3% mIoU, as presented in Table 3.

Moreover, the minimum detectable crack width per pixel is contingent upon factors such as camera resolution, focal length, sensor type, lens characteristics, including the performance of our segmentation model and the characteristics of the data used for training the STRNet. Our test dataset labels were meticulously annotated down to individual pixels on high-resolution images (2992 × 2000 × 3), facilitating precise crack detection. Nonetheless, determining the minimum detectable crack width at significantly greater distances

Table 3. Comparative studies: STRNet and STRNet + TTA.

Cases	Images size	Image number	mIoU (%)
STRNet (Training)	1024 × 512	1203	—
STRNet (Test)	1024 × 512, 1280 × 720	545	92.6
STRNet (Test on new structure)	2992 × 2000	11	80
STRNet + TTA (Test on new structure)	2992 × 2000	132	90.3

STRNet: semantic transformer representation network; TTA: test time augmentation.

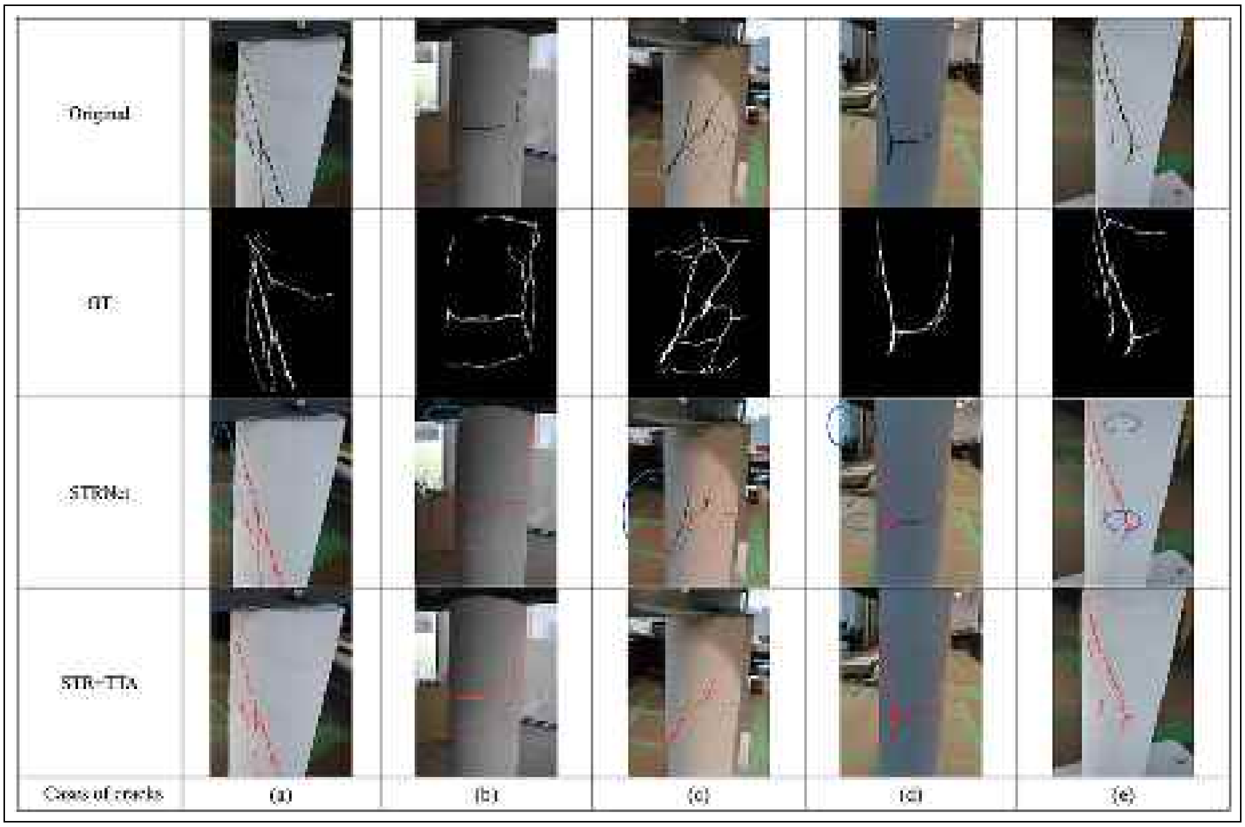


Figure 17. Crack segmentation results using STRNet and STRNet + TTA.
STRNet: semantic transformer representation network; TTA: test time augmentation.

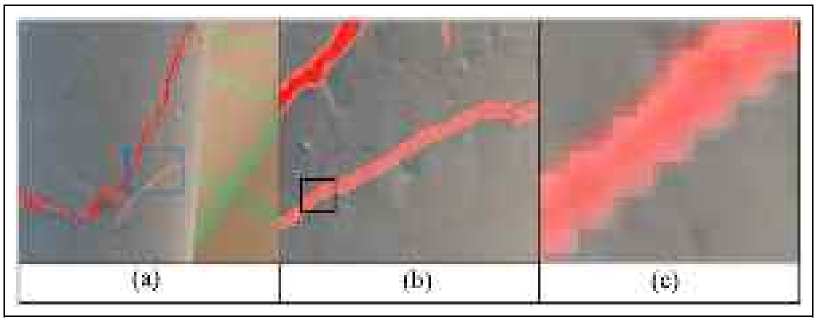


Figure 18. Progressive zoom levels of crack segmentation.

was beyond the scope of this study, as it would adversely affect the quality of the 3D reconstruction, which is the primary focus of the research. We determined that maintaining a camera distance of 1 to 2 m provides an optimal compromise between detailed crack detection and high-quality 3D reconstructions.

Figure 18 demonstrates the precision of pixel-level crack segmentation through progressively magnified views of the same area on a column. Each image in the sequence captures increasingly detailed aspects of the segmented cracks.

3D reconstruction and damage mapping using Nerfacto

The NeRF and Nerfacto techniques introduced in section "Data" are available within the Nerfstudio framework[45] to facilitate 3D reconstruction. Nerfstudio is a modular PyTorch framework known for its extensibility and versatility in neural radiance field development. It consolidates various NeRF techniques into reusable, modular components, enabling real-time visualization of 3D reconstructed scenes, offering comprehensive

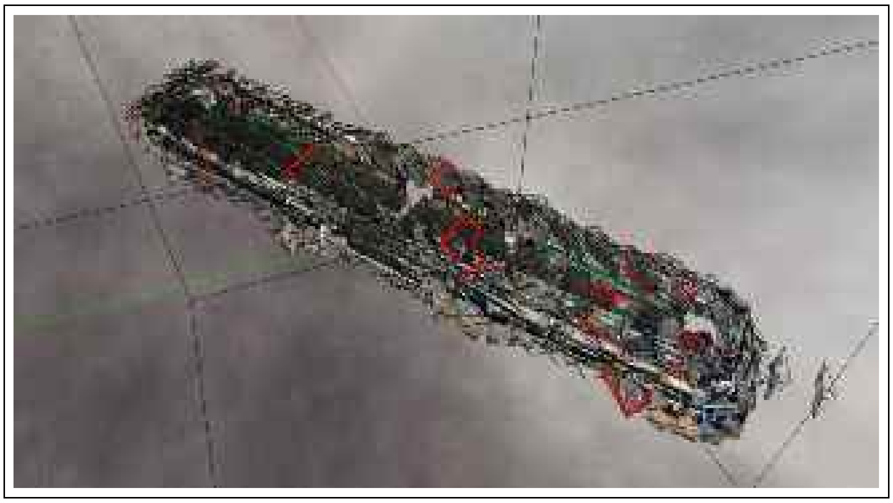


Figure 19. Global distributions of all input images based on the estimated camera poses using intrinsic and extrinsic parameters.

controls, and providing a user-friendly, end-to-end workflow for creating diverse NeRF-based models from user-captured data. As an open-source project, Nerfstudio benefits from ongoing enhancements contributed by both academic and industry experts.

Through trials, we found that the original NeRF model was not suitable for our structural case due to its rendering limitations, including excessively high computational costs, which resulted in excessively long training times for the DNNs within NeRF, ultimately causing the training process to be halted. Therefore, in this paper, we extensively investigated Nerfacto through various parametric studies, including modifications to its DNNs and methods of collecting RGB image data, to improve 3D reconstruction and damage mapping.

As an initial step, the collected RGB input images were fed into the Nerfstudio framework. It rendered the global distributions of the entire RGB input images, as presented in Table 2, with their corresponding global locations determined from the intrinsic and extrinsic parameters that were extracted from Metashape, as shown in Figure 19.

Using the default hyperparameter settings, the web URL provides a real-time rendering of the model's training progress. Initial renderings appear blurry, as shown in Figure 20(a), but gradually become clearer as the model's color training progresses, as shown in Figure 19.

Parametric studies of Nerfacto. In this section, core evaluation metrics used for the evaluation of the 3D reconstructed model are introduced, along with various parametric studies, including modifications to its DNNs and methods of collecting RGB input image data, to examine the effects on the quality of the 3D reconstruction.

Evaluation metrics. To evaluate the results of the 3D reconstruction through the extensive parametric studies, several evaluation metrics were adopted. These included the MSE loss function, Peak Signal to Noise Ratio (PSNR), and Structural Similarity Index (SSIM), which are essential for assessing the quality of the rendered images.[45]

PSNR measures the ratio between the maximum possible signal value and the distorting noise affecting its representation, providing insights into the fidelity of the reconstruction. It is calculated using Equation (29):

$$\mathrm{PSNR} = 10\log_{10}(R^2/\ \mathrm{MSE}), \text{where } R\ =\ 255\,dB \quad (29)$$

SSIM, on the other hand, is a method for predicting the perceived quality of images, considering factors like luminance, contrast, and structure. SSIM is calculated using Equation (30). Both PSNR and SSIM are popular metrics in the field; however, each metric has its limitations. Therefore, employing all three metrics, including MSE Loss, offers a more comprehensive assessment of our 3D reconstruction quality.

$$\mathrm{SSIM}(x,\ y) = \frac{\left(2\mu_x\mu_y + c_1\right)\left(2\sigma_{xy} + c_2\right)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)}, \quad (30)$$

where μ_x is the pixel sample mean of x, μ_y is the pixel sample mean of y, σ_x^2 is the variance of x, σ_y^2 is the variance of y, σ_{xy} is the covariance of x and y, $c_1 = (k_1L)^2$,

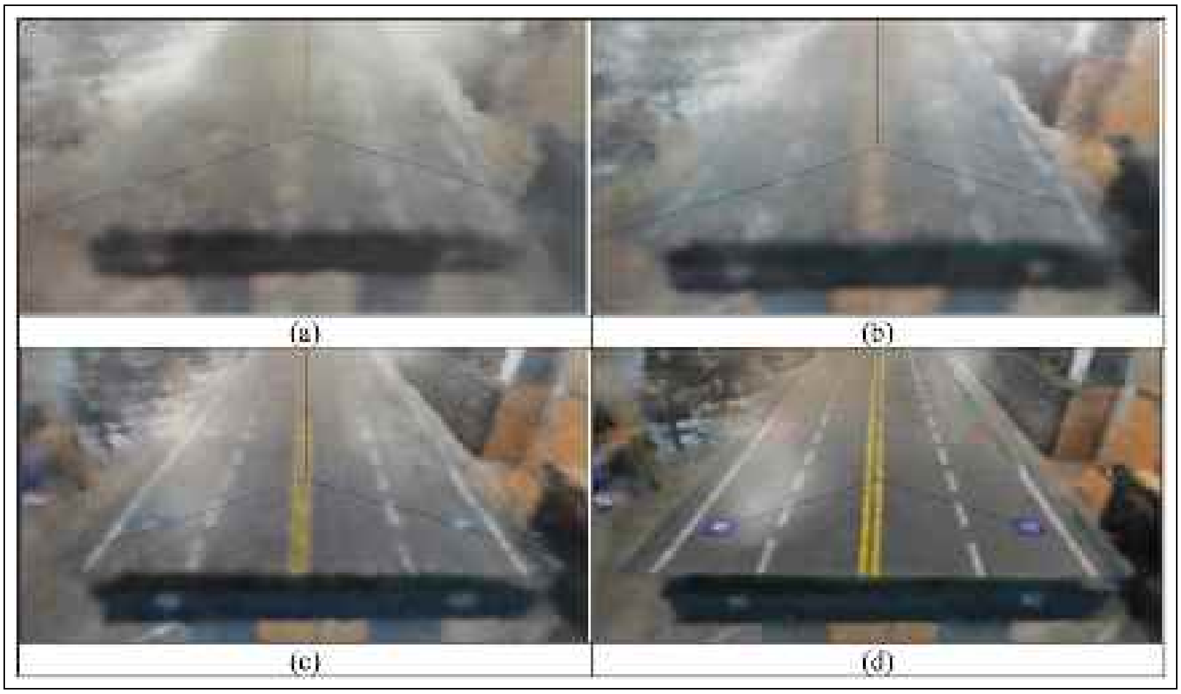


Figure 20. Real-time training progress of Nerfacto is displayed on a web URL. (a) Iteration 300. (b) Iteration 700. (c) Iteration 1000 and (d) Iteration 2000.

$c_2 = (k_2 L)^2$, L is the dynamic range of the pixel values ($2^{\text{bits per pixel}} - 1$), $k_1 = 0.01$, and $k_2 = 0.03$.

PSNR measures the peak error between the reconstructed and original images, providing a quantitative measure of the reconstruction's precision in terms of pixel intensity errors. This metric is crucial for understanding the technical accuracy of the model's output relative to the ground truth. Conversely, SSIM evaluates the visual perception of the reconstructed images by analyzing changes in luminance, contrast, and structure compared to the original images. This metric offers insights into the perceptual impact of our reconstruction process, addressing aspects of image quality that PSNR does not capture, such as visual fidelity and textural preservation.

The use of both metrics is standard in NeRF-related research because they together offer a comprehensive overview of both the objective accuracy and subjective quality of the reconstructed images.

However, as Tancik et al.[45] point out, these evaluations are typically based on images that follow the same capture trajectory as the images used for training. This approach can be misleading, especially in real-world applications that capture data from unstructured environments and render new views with significant deviations from the training set. Relying solely on PSNR and SSIM might not fully reflect the model's performance in diverse and practical scenarios.

It is important to recognize that while metrics such as PSNR offer some insights, they do not encompass a complete understanding of a model's performance, particularly for views that diverge substantially from the capture trajectory. A more holistic approach involves qualitative evaluation, such as using an interactive viewer, which can provide a deeper and more comprehensive understanding of the model's capabilities in various conditions. Despite these considerations, quantitative metrics still hold value in certain aspects of our analysis.

Parametric studies including modification of DNNs. In this section, various parametric studies were conducted, and the performance of the Nerfacto was compared to some existing 3D reconstruction methods, including the original NeRF. As an example of a traditional 3D reconstruction method, a dense point cloud and mesh-based method using photogrammetry software was implemented. This method was presented as Case 1 (C1) in the results table (Table 4). For Case 2 (C2), the original NeRF model was implemented. In Case 3 (C3), the effects of relatively lower input image resolution were investigated. For Case 4 (C4), higher resolution with a relatively smaller number of input images was also investigated. In Case 5 (C5), the methods of collecting input images were investigated. In this case,

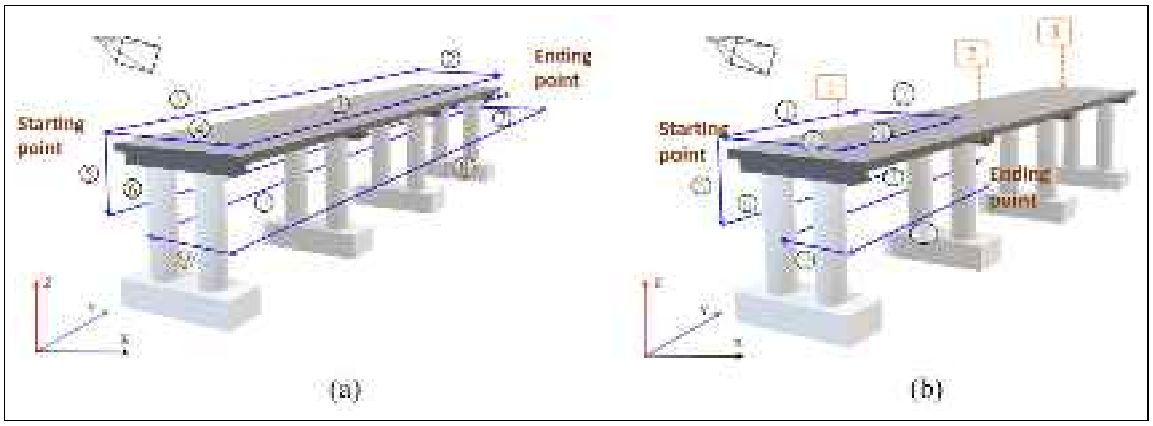


Figure 21. Photography routes. (a) Entire three-span route of photography and (b) each span route of photography.

Table 4. Comparative and parametric studies.

Case (C3–C7: Nerfacto)	Image size (pixel)	Number of images	PSNR (db)	SSIM	Loss function (MSE)	Training time (hour)
C1: Photogrammetry	2992 × 2000	1931	—	—	—	—
C2: Original NeRF	2992 × 2000	1931	—	—	—	More than a month
C3: Low resolution	1024 × 720	1931	21.883	0.7724	0.00813	0.53
C4: High resolution, small number of input images	2992 × 2000	743	21.115	0.7382	0.00911	1.13
C5: Collecting images along the entire three-span route	2992 × 2000	1931	22.660	0.8167	0.00610	1.38
C6: Collecting images span by span	2992 × 2000	1931	23.119	0.8029	0.00603	1.37
C7: Collecting images span by span with changed hyperparameters	2992 × 2000	1931	26.570	0.8025	0.00278	3.87

PSNR: peak signal to noise ratio; MSE: mean squared error.

two different methods of collecting the input images were implemented, as shown in Figure 21.

As illustrated in Figure 21, the three bridge deck spans were scanned along the entire three-span route by moving back and forth, with the images assigned numbered starting and ending points. Another method of collecting input images, Case 6 (C6) in the results table (Table 4), was to scan each span separately. In these two image collection methods, the same total number of 1931 images was used, and efforts were made to capture the images at similar locations and angles to isolate the effects of the collection order. The final case, Case 7 (C7), used the span-by-span image collection method, which was selected based on the results and additional parametric studies of the Nerfacto model. The final results for this case are reported in Table 4.

As presented in Table 4, the traditional photogrammetry-based method has fundamentally different mechanisms for the final 3D reconstruction, so the evaluation metrics used for the Nerfacto model cannot be directly applied. Therefore, the final result of the photogrammetry-based method is presented only in Figure 22. The original NeRF was also implemented using the same input image data, but due to its high computational requirements, the simulation was stopped after 20 days of training. We concluded that the original NeRF cannot be effectively implemented for large-scale civil infrastructure problems.

From C3 to C7, the Nerfacto model was investigated extensively. In C3, relatively lower-resolution input images were used, resulting in degraded evaluation metric values compared to C4–C7. This suggests that high-resolution input images are beneficial for creating high-quality 3D reconstructed models. In C4, high-resolution images with a much smaller number of input images were used to train the Nerfacto, and the results were still slightly better than those of C3 with a much greater number of input images. Through C5 and C6, we found that the order of image collection affects the final 3D reconstruction result. The conclusion is that input images should be collected in a grid or segment-based manner to help the embedded DNNs learn the geometry of the 3D spatial structure.

As the final case study (C7), the Nerfacto model was modified with deeper DNN-A and DNN-B networks, more training iterations, a larger training batch size in terms of the number of rays processed, and a greater number of 3D sample points per ray using uniform random sampling, as presented in Figures 8 and 9. Additionally, the number of 3D sample points per ray for the proposal sampling and DNN-A was increased, as shown in Figure 9. Therefore, the modified Nerfacto model demonstrated the best performance in terms of the evaluation metrics presented in Table 4, including PSNR, SSIM, and MSE. All the detailed final hyperparameters of the Nerfacto model for the final result of C7 are presented in Table 5.

The visual results of these comparative and parametric studies are presented in Figure 22. As shown in these figures, the traditional photogrammetry-based approach does not provide meaningful results in terms of resolution. The generated 3D model is too blurry and coarse to discern the structural details. The results of C4 and C5 are also not significantly different from those of C1. In contrast, C6 and C7 provide much better results with higher resolution and clearer definition of the structural members. Among these two, the result of C7 is superior in terms of the evaluation metrics presented in Table 4 and the overall visual quality. Based on these results, both in terms of the evaluation metrics and the subjective visual assessment, C7 was selected as the final Nerfacto model.

The finally selected Nerfacto model, with modified DNN architectures and adjusted hyperparameters, generated a high-definition 3D reconstruction of the entire bridge, as presented in Figure 23. The key benefits of this Nerfacto-based 3D reconstruction are that it provides a high-definition 3D model and can generate 3D views from any angle and location in real-time, optimized through the training of the DNNs. In contrast, the traditional 3D models generated by photogrammetry methods have several disadvantages. They are time-consuming to render and are limited to a single, large 3D model. Additionally, they do not allow for the generation of new 3D models from different angles and locations, which can be inconvenient for users.

3D damage mapping. Through the various case studies in the previous section, we found that the Nerfacto approach can generate high-resolution, clear 3D reconstructions of a large-scale bridge system. Another objective of this article is to perform 3D damage mapping of the segmented damage using deep learning models for structural damage detection. As an example, we adopted the STRNet with TTA module from the previous section "STRNet + TTA evaluation."

To generate a 3D model with the detected damage segmented, there are two different approaches that can be used for extracting the intrinsic and extrinsic parameters for SfM, which are then used for 3D model generation using Nerfacto. Therefore, we conducted two different case studies:

Case A: In this case, we used the raw RGB input images to extract the intrinsic and extrinsic parameters for SfM, and then generated the 3D model using the intrinsic and extrinsic parameters from the raw images, combined with the damage segmented images from the STRNet with TTA.

Case B: In this case, we used the damage-segmented images from the STRNet with TTA to extract the intrinsic and extrinsic parameters for SfM, and then generated the 3D model using the intrinsic and extrinsic parameters from the damaged images. These two cases are well illustrated in Figure 24 for better understanding.

These two different approaches were compared using only 85 input images for simplified visualization, and Figure 25 illustrates the results for a specific iteration and batch. In Case A, the input pictures are well-aligned, indicating that the camera pose estimation achieved through extracting intrinsic and extrinsic parameters from the raw input images for 3D reconstruction using Nerfacto was accurate. In contrast, Case B reveals input images that are not as well aligned, suggesting that the camera pose estimation based on the damage-segmented images was less accurate.

The reason for the coarse alignment of the input batch images in Case B is that the cracks were not perfectly segmented. Therefore, during the extraction of intrinsic and extrinsic parameters using the crack-segmented images, the same objects in the input images were not accurately indicated at the same locations and angles within the parameters. This misalignment, resulting from inaccurately aligned images, leads to incorrect ray tracing from each image, which reduces the quality of the 3D reconstruction. These outcomes were observed during our tests with a set of 85 images focused on reconstructing a pair of bridge columns.

While slight variations in crack segmentation in the SfM stage can lead to potentially severe misalignment of images, this issue can be neglected during the training phase of the Nerfacto model. Nerfacto, a neural network-based model, is designed to learn from and

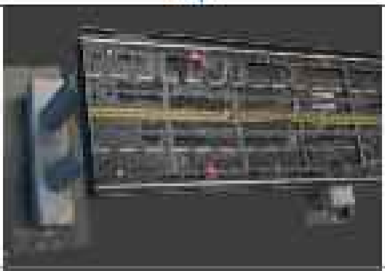
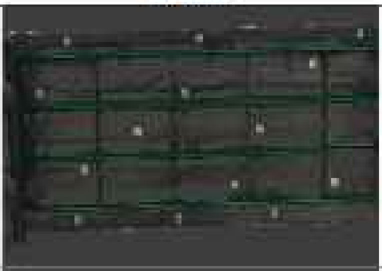
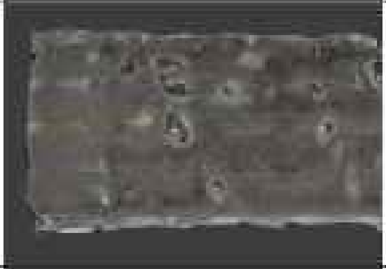
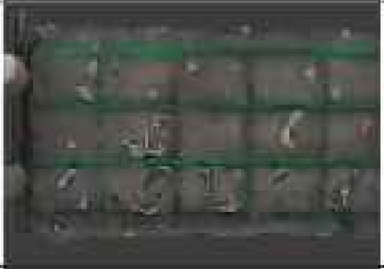
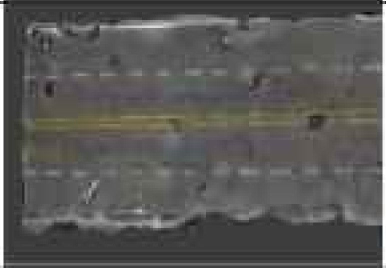
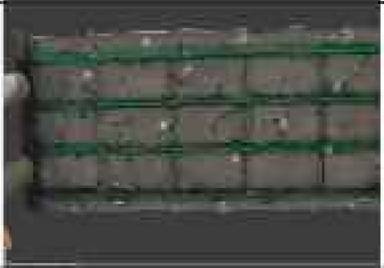
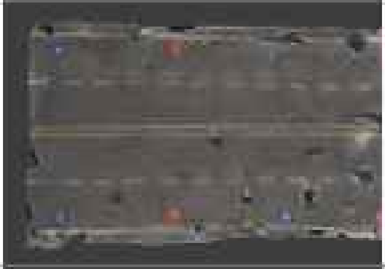
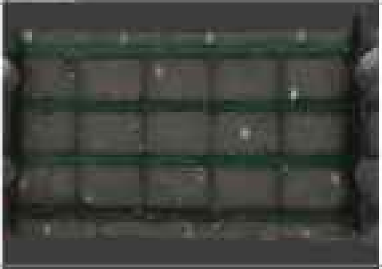
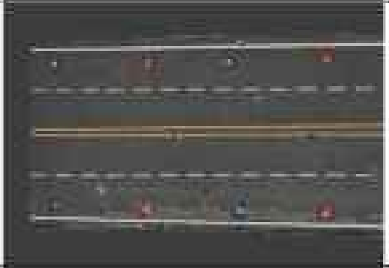
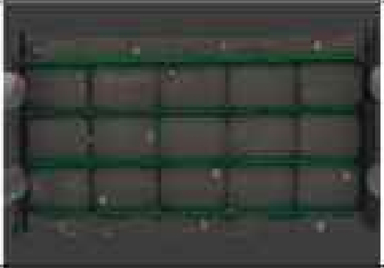
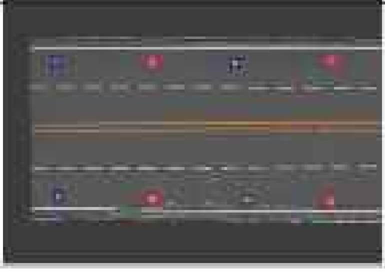
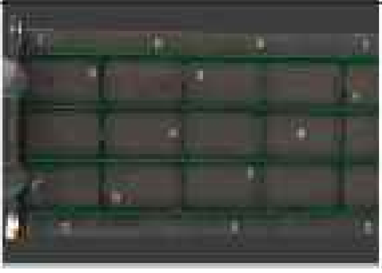
	Top	Bottom
C1: Photogrammetry • Number of images: 1,931 • Resolution: 2992×2000 • Entire three-span route		
C3: Nerfacto • Number of images: 1,931 • Resolution: 1024×720 • Entire three-span route • Default hyperparameters		
C4: Nerfacto • Number of images: 743 • Resolution: 2992×2000 • Entire three-span route • Default hyperparameters		
C5: Nerfacto • Number of images: 1,931 • Resolution: 2992×2000 • Entire three-span route • Default hyperparameters		
C6: Nerfacto • Number of images: 1,931 • Resolution: 2992×2000 • Each span route • Default hyperparameters		
C7: Modified Nerfacto • Number of images: 1,931 • Resolution: 2992×2000 • Each span route • Hyperparameters changed		

Figure 22. The 3D reconstruction results of each case.

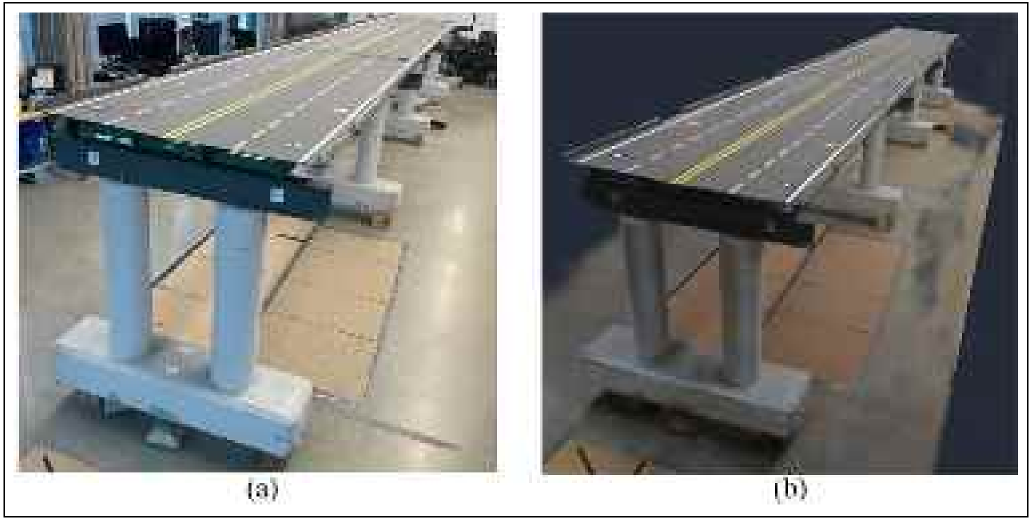


Figure 23. Comparison with an RGB image and Nerfacto-based 3D reconstruction. (a) Image of the bridge model and (b) 3D Reconstructed model of the bridge model.

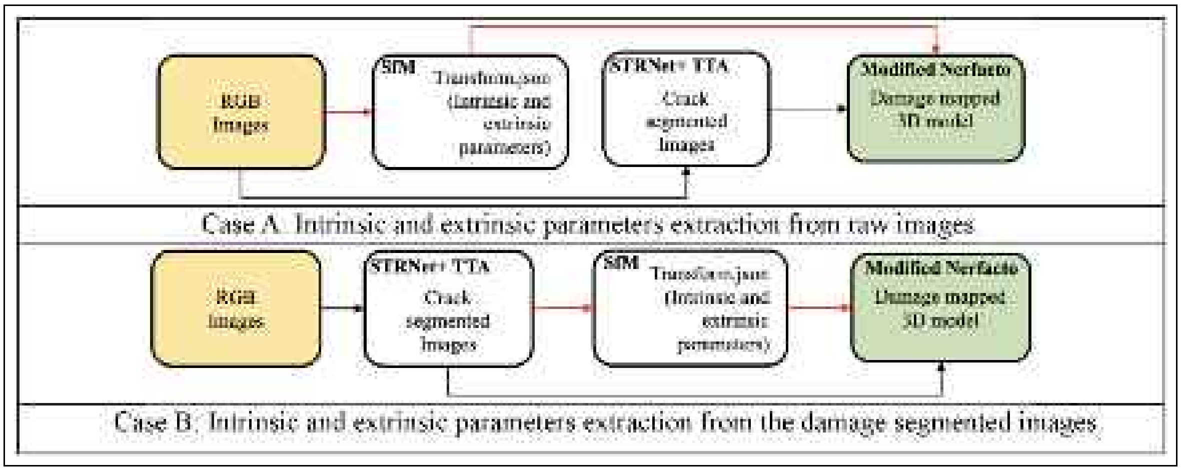

RGB Images
SfM
Transform.json (Intrinsic and extrinsic parameters)
STRNet+ TTA
Crack segmented Images
Modified Nerfacto
Damage mapped 3D model
Case A. Intrinsic and extrinsic parameters extraction from raw images
RGB Images
STRNet+ TTA
Crack segmented Images
SfM
Transform.json (Intrinsic and extrinsic parameters)
Modified Nerfacto
Damage mapped 3D model
Case B: Intrinsic and extrinsic parameters extraction from the damage segmented images

Figure 24. Two different approaches for extracting intrinsic and extrinsic parameters for 3D model generation. Case A: Intrinsic and extrinsic parameters extraction from raw images and Case B: Intrinsic and extrinsic parameters extraction from the damage-segmented images.

Table 5. Hyperparametric settings for modified Nerfacto of C7.

Hyper parameters	Default setting	Our setting
Number of iterations for training	30,000	100,000
Number of rays (pixels) processed per training batch	4096	8192
Number of 3D sample points per ray for proposal sampling	256 → 96 → 48	256 → 128 → 64
Number of 3D sample points per ray for main DNN-A	48	64
Hidden layers' dimension of main DNN-A.	64	256
Hidden layers' dimension of main DNN-B.	64	128

DNN: deep neural network.

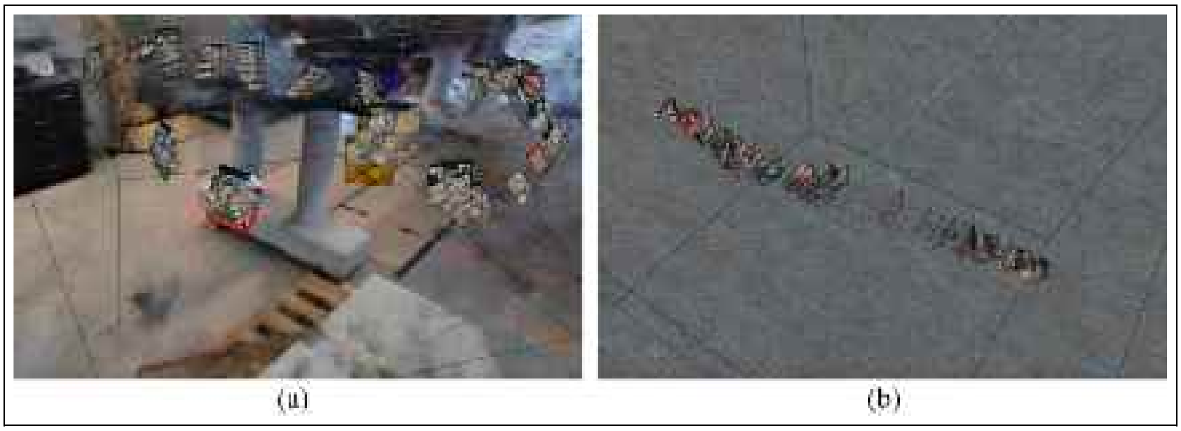


Figure 25. Images aligned for the camera pose estimation case. (a) Result of Case A and (b) Result of Case B.

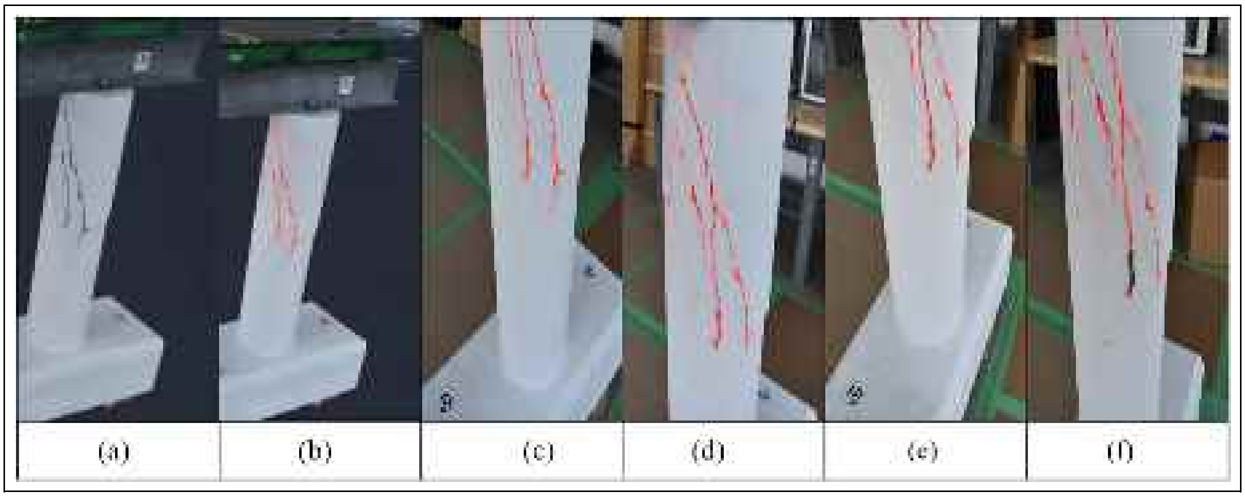


Figure 26. 3D damage mapping consistency across varied 2D crack segmentation images. (a) Base 3D reconstruction. (b) Damage mapped 3D reconstruction. (c) Training image 1. (d) Training image 2. (e) Training image 3 and (f) Training image 4.

average these variations in segmentation. The model's training process involves optimizing network weights based on a broad set of training data, effectively allowing it to discern and emphasize the most common features in the data. Therefore, as long as the majority of the images correctly depict the segmented cracks, the model can accurately map the cracks onto the 3D structure. The example of this capacity to average out errors advantage is illustrated in Figure 26.

The 2D training images (1–4) exhibit slight variations in crack segmentation. However, the final result of the damage mapped 3D reconstructed model's cracks shown in Figure 26(b) represents the overall average of the segmented cracks obtained during the training process. The model learns from all four training images and presents the average (i.e., predominant) values for the same area (e.g., Crack) of the structure. Image (a) displays the 3D reconstruction of a bridge pier without any damage segmentation, serving as a baseline for comparison.

Therefore, in Case A, we generated a 3D damage map using the modified Nerfacto approach with the same hyperparameters and number of hidden layers as the C7 model in Table 4. The generated 3D damage map is presented in Figure 27. This step aims to accomplish size quantification and intuitive localization of the damage on the bridge model. By visually representing the damage in a 3D context, we were able to gain a more comprehensive and realistic understanding of the structural integrity of the bridge.

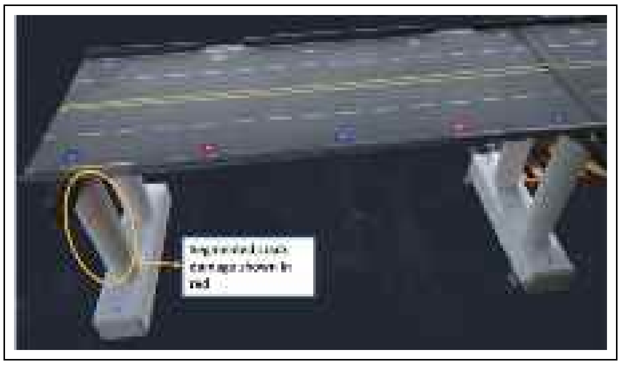


Figure 27. 3D mapping of the segmented cracks.

A key highlight of our results is showcased in Figure 27, which presents an image of the segmented cracks mapped onto the 3D bridge model. This visual representation illustrates the accurate localization and depiction of the extent of damage to the bridge structure. The segmented cracks are clearly delineated, providing an intuitive understanding of the damage's location and severity. Furthermore, Figure 28 offers a comparative visualization between unsegmented and segmented cracks on the 3D model. This comparison vividly demonstrates the increased clarity and detail achieved through our segmentation process. The segmented cracks are distinctly visible, offering a more precise and informative view of the structural damage compared to the unsegmented version.

Conclusion

This article presents an innovative approach to generate high-definition 3D reconstructed models for bridge systems and to map detected (segmented) damage within the reconstructed 3D model using a deep learning-based damage identification approach with computer vision input. The 3D reconstruction method involves an advanced NeRF-based approach, specifically a modified Nerfacto model. For damage segmentation, the state-of-the-art STRNet with TTA was developed. The main finding of this paper is that this DNN-based 3D reconstruction method, including damage mapping, works well for large-scale civil infrastructure. The proposed method (modified Nerfacto) effectively learns various features of the structure's appearance from different angles and locations using embedded DNNs. These developments enable more holistic damage mapping for actual civil infrastructures, facilitating more systematic and efficient monitoring and management. Detailed findings from extensive case studies include:

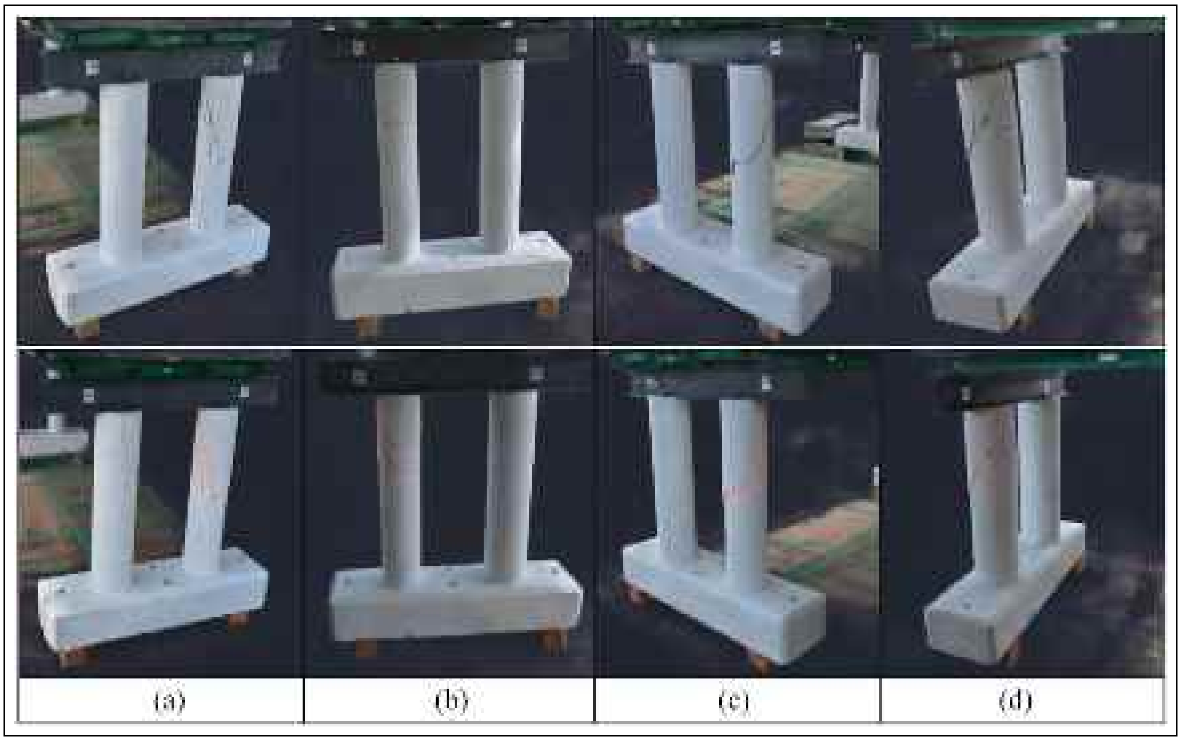


Figure 28. Visualization of various angle views of the segmented cracks on the 3D model. (a) Front view. (b) Rear view. (c) Front side view and (d) Rear side view.

(1) Traditional photogrammetry-based 3D reconstruction did not provide any meaningful results.
(2) The original NeRF model did not perform well for the structure of interest due to its excessive computational power and cost requirements.
(3) The Nerfacto model incorporates advanced methods for selecting a possible number of 3D sample points and generating inputs for the DNNs, such as multiresolution hash encoding, spherical harmonics encoding, and appearance embedding. These methods enable high-definition 3D reconstruction of the structure by reducing computational cost and allowing efficient training of the embedded DNNs.
(4) Existing DNNs' hidden layers, batch size, 3D sample points per ray for uniform random sampling, and 3D sample points per ray for proposal sampling of the Nerfacto need to be significantly increased to learn the various features of the large-scale structure, which led to the modified Nerfacto.
(5) The manner of collecting RGB inputs is critical for generating a high-definition 3D model. In our case, grid-style collection of input images is more beneficial for efficiently establishing 3D structural information.
(6) Raw images should be used to extract intrinsic and extrinsic parameters for camera location and angle information for 3D damage mapping. This is crucial for generating a 3D reconstructed model with segmented damage information.

For future work, the proposed modified Nerfacto model can be applied to real-world infrastructure located outdoors, which involves larger scales and exposure to various lighting conditions. As for image collection methods, both manual and autonomous UAVs can be considered.[31]

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The research was supported by the Mitacs L2M program IT34014, Research Manitoba Proof-of-Concept Grant (4914), NSERC Discovery Grant (RGPIN-2022-04120) and the CFI JELF grant, (37394).

ORCID iD

Youngjin Cha https://orcid.org/0000-0002-0738-5615

References

1. Cha YJ, Ali R, Lewis J, et al. Deep learning-based structural health monitoring. *Autom Constr* 2024; 161: 105328.
2. Hammad AW, Yan J and Mostofi B. Recent development of bridge management systems in Canada. In: *2007 Annual Conference and Exhibition of the Transportation Association of Canada: Transportation-An Economic Enabler* (Les Transports: Un Levier Economique) Transportation Association of Canada (TAC), 2007.
3. Gaudreault V and Lemire P. *The age of public infrastructure in Canada*. Ottawa, ON: Statistics Canada, 2006.
4. Giurgiutiu V. *Structural health monitoring of aerospace composites*. London: Elsevier, 2015.
5. Deraemaeker A, Preumont A, Reynders E, et al. Vibration-based structural health monitoring using large sensor networks. *Smart Struct Syst* 2010; 6(3): 335–347.
6. Cha YJ and Buyukozturk O. Structural damage detection using modal strain energy and hybrid multiobjective optimization. *Comput-Aided Civ Infrastruct Eng* 2015; 30(5): 347–358.
7. Sanchez-Silva M, Klutke GA and Rosowsky DV. Life-cycle performance of structures subject to multiple deterioration mechanisms. *Struct Saf* 2011; 33(3): 206–217.
8. Al-Mahaidi R and Kalfat R. *Rehabilitation of concrete structures with fiber-reinforced polymer*. Oxford: Butterworth-Heinemann, 2018.
9. Cha YJ, You K and Choi W. Vision-based detection of loosened bolts using the Hough transform and support vector machines. *Autom Construct* 2016; 71: 181–188.
10. Feng D and Feng MQ. Computer vision for SHM of civil infrastructure: from dynamic response measurement to damage detection–a review. *Eng Struct* 2018; 156: 105–117.
11. Zakeri H, Nejad FM and Fahimifar A. Image based techniques for crack detection, classification and quantification in asphalt pavement: a review. *Arch Comput Methods Eng* 2017; 24: 935–977.
12. Kheradmandi N and Mehranfar V. A critical review and comparative study on image segmentation-based techniques for pavement crack detection. *Constr Build Mater* 2022; 321: 126162.
13. Cha YJ, Choi W and Büyüköztürk O. Deep learning-based crack damage detection using convolutional neural networks. *Comput-Aided Civ Infrastruct Eng* 2017; 32(5): 361–378.
14. Cha YJ, Choi W, Suh G, et al. Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. *Comput-Aid Civ Infrastruct Eng* 2018; 33(9): 731–747.
15. Kang D and Cha YJ. Autonomous UAVs for structural health monitoring using deep learning and an ultrasonic beacon system with geo-tagging. *Comput-Aid Civ Infrastruct Eng* 2018; 33(10): 885–902.
16. Ronneberger O, Fischer P and Brox T. U-net: Convolutional networks for biomedical image segmentation. In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th International Conference,*

Munich, Germany, 5–9 October 2015, Proceedings, Part III 18, pp. 234–241. Springer International Publishing.

17. Jenkins MD, Carr TA, Iglesias MI, et al. A deep convolutional neural network for semantic pixel-wise segmentation of road and pavement surface cracks. In: *2018 26th European signal processing conference (EUSIPCO)*, Rome, Italy, 2018, pp. 2120–2124. IEEE.
18. Liu Z, Cao Y, Wang Y, et al. Computer vision-based concrete crack detection using U-net fully convolutional networks. *Autom Constr* 2019; 104: 129–139.
19. Fu H, Meng D, Li W, et al. Bridge crack semantic segmentation based on improved Deeplabv3+. *J Marine Sci Eng* 2021; 9(6): 671.
20. Choi W and Cha YJ. SDDNet: real-time crack segmentation. *IEEE Trans Ind Electron* 2019; 67(9): 8016–8025.
21. Kang DH and Cha YJ. Efficient attention-based deep encoder and decoder for automatic crack segmentation. *Struct Health Monit* 2022; 21(5): 2190–2205.
22. Dryden N, Maruyama N, Benson T, et al. Improving strong-scaling of CNN training by exploiting finer-grained parallelism. In: *2019 IEEE International parallel and distributed processing symposium (IPDPS)*, Rio de Janeiro, Brazil, 2019, pp. 210–220. IEEE.
23. Ye C, Butler L, Calka B, et al. *A digital twin of bridges for structural health monitoring*. Stanford, CA: Stanford University, 2019.
24. Teng S, Chen X, Chen G, et al. Structural damage detection based on transfer learning strategy using digital twins of bridges. *Mech Syst Signal Process* 2023; 191: 110160.
25. Xiang C, Gan VJ, Guo J, et al. Semi-supervised learning framework for crack segmentation based on contrastive learning and cross pseudo supervision. *Measurement* 2023; 217: 113091.
26. Hoshyarmanesh H, Abbasi A, Moein P, et al. Design and implementation of an accurate, portable, and time-efficient impedance-based transceiver for structural health monitoring. *IEEE/ASME Trans Mechatron* 2017; 22(6): 2809–2814.
27. Ayhan MS and Berens P. Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks. In: *Medical imaging with deep learning*, 2018.
28. Wang G, Li W, Aertsen M, et al. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing* 2019; 338: 34–45.
29. Zhao S, Kang F, Li J, et al. Structural health monitoring and inspection of dams based on UAV photogrammetry with image 3D reconstruction. *Autom Construt* 2021; 130: 103832.
30. Ali R, Kang D, Suh G, et al. Real-time multiple damage mapping using autonomous UAV and deep faster region-based neural networks for GPS-denied structures. *Autom Constr* 2021; 130: 103831.
31. Waqas A, Kang D and Cha YJ. Deep learning-based obstacle-avoiding autonomous UAVs with fiducial marker-based localization for structural health monitoring. *Struct Health Monit* 2024; 23(2): 971–990.
32. Ye XW, Jin T and Yun CB. A review on deep learning-based structural health monitoring of civil infrastructures. *Smart Struct Syst* 2019; 24(5): 567–585.
33. Chen C, Fu H, Zheng Y, et al. The advance of digital twin for predictive maintenance: the role and function of machine learning. *J Manufactur Syst* 2023; 71: 581–594.
34. Kong X and Hucks RG. Preserving our heritage: a photogrammetry-based digital twin framework for monitoring deteriorations of historic structures. *Autom Constr* 2023; 152: 104928.
35. Hartley R and Zisserman A. *Multiple view geometry in computer vision*. Cambridge: Cambridge University Press, 2003.
36. Schonberger JL and Frahm JM. Structure-from-motion revisited. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, Las Vegas, Nevada, 2016, pp. 4104–4113.
37. Xue Y, Shi P, Jia F, et al. 3D reconstruction and automatic leakage defect quantification of metro tunnel based on SfM-deep learning method. *Underground Space* 2022; 7(3): 311–323.
38. Chaiyasarn K, Buatik A, Mohamad H, et al. Integrated pixel-level CNN-FCN crack detection via photogrammetric 3D texture mapping of concrete structures. *Autom Constr* 2022; 140: 104388.
39. Remondino F, Barazzetti L, Nex F, et al. UAV photogrammetry for mapping and 3d modeling–current status and future perspectives. *Int Arch Photogramm Remote Sens Spat Inform Sci* 2012; 38: 25–31.
40. Schenk T, Seo S and Csathó B. Accuracy study of airborne laser scanning data with photogrammetry. *Int Arch Photogramm Remote Sens* 2001; 34(part 3): W4.
41. Mildenhall B, Srinivasan PP, Tancik M, et al. NeRF: Representing scenes as neural radiance fields for view synthesis. *Commun ACM* 2021; 65(1): 99–106.
42. Gao K, Gao Y, He H, et al. Nerf: Neural radiance field in 3d vision, a comprehensive review. *arXiv preprint arXiv:2210.00379*, 2022.
43. Pal A, Lin JJ, Hsieh SH, et al. Activity-level construction progress monitoring through semantic segmentation of 3D-informed orthographic images. *Autom Constr* 2024; 157: 105157.
44. Yu Z, Shen Y, Zhang Y, et al. Automatic crack detection and 3D reconstruction of structural appearance using underwater wall-climbing robot. *Autom Constr* 2024; 160: 105322.
45. Tancik M, Weber E, Ng E, et al. 2023. Nerfstudio: a modular framework for neural radiance field development. In: *ACM SIGGRAPH 2023 conference proceedings*, Los Angeles, CA, pp. 1–12.
46. Kim G and Cha Y (2024). 3D Pixelwise damage mapping using a deep attention based modified Nerfacto. *Autom Constr* 2024; 168: 105878.
47. Agisoft Metashape Professional Edition (version 2.1), https://www.agisoft.com (2024, accessed 5 November 2024).

48. Avenash R and Viswanath P. Semantic segmentation of satellite images using a modified CNN with Hard-Swish activation function. *In: VISIGRAPP (4: VISAPP)*, Prague, Czech Republic, 2019, pp. 413–420.
49. Ioffe S and Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: *International conference on machine learning*, Lille, France, 2015, pp. 448–456. PMLR.
50. Barron JT, Mildenhall B, Verbin D, et al. Mip-NeRF 360: unbounded anti-aliased neural radiance fields. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, New Orleans, Louisiana, 2022, pp. 5470–5479.
51. Müller T, Evans A, Schied C, et al. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans Graphics* 2022; 41(4): 1–15.
52. Verbin D, Hedman P, Mildenhall B, et al. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In: *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, New Orleans, Louisiana, 2022, pp. 5481–5490. IEEE.
53. Martin-Brualla R, Radwan N, Sajjadi MS, et al. NeRF in the wild: Neural radiance fields for unconstrained photo collections. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 7210–7219.