Zhang, Yunfan; Scargill, Tim; Vaishnav, Ashutosh; Premsankar, Gopika; Di Francesco, Mario; Gorlatova, Maria

InDepth: Real-time Depth Inpainting for Mobile Augmented Reality

Published in:
Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies

DOI:
10.1145/3517260

Published: 01/03/2022

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Zhang, Y., Scargill, T., Vaishnav, A., Premsankar, G., Di Francesco, M., & Gorlatova, M. (2022). InDepth: Real-time Depth Inpainting for Mobile Augmented Reality. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *6*(1), 1-25. Article 37. https://doi.org/10.1145/3517260

InDepth: Real-time Depth Inpainting for Mobile Augmented Reality

YUNFAN ZHANG, Duke University, USA
TIM SCARGILL, Duke University, USA
ASHUTOSH VAISHNAV, Aalto University, Finland
GOPIKA PREMSANKAR, University of Helsinki, Finland
MARIO DI FRANCESCO, Aalto University, Finland
MARIA GORLATOVA, Duke University, USA

Mobile Augmented Reality (AR) demands realistic rendering of virtual content that seamlessly blends into the physical environment. For this reason, AR headsets and recent smartphones are increasingly equipped with Time-of-Flight (ToF) cameras to acquire depth maps of a scene in real-time. ToF cameras are cheap and fast, however, they suffer from several issues that affect the quality of depth data, ultimately hampering their use for mobile AR. Among them, scale errors of virtual objects – appearing much bigger or smaller than what they should be – are particularly noticeable and unpleasant. This article specifically addresses these challenges by proposing InDepth, a real-time depth inpainting system based on edge computing. InDepth employs a novel deep neural network (DNN) architecture to improve the accuracy of depth maps obtained from ToF cameras. The DNN fills holes and corrects artifacts in the depth maps with high accuracy and eight times lower inference time than the state of the art. An extensive performance evaluation in real settings shows that InDepth reduces the mean absolute error by a factor of four with respect to ARCore DepthLab. Finally, a user study reveals that InDepth is effective in rendering correctly-scaled virtual objects, outperforming DepthLab.

CCS Concepts: • **Human-centered computing** → **Mixed / augmented reality**; **Mobile devices**.

Additional Key Words and Phrases: Augmented reality, depth sensing, depth inpainting, edge computing, user experience

ACM Reference Format:
Yunfan Zhang, Tim Scargill, Ashutosh Vaishnav, Gopika Premsankar, Mario Di Francesco, and Maria Gorlatova. 2022. InDepth: Real-time Depth Inpainting for Mobile Augmented Reality. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 1, Article 37 (March 2022), 25 pages. https://doi.org/10.1145/3517260

1 INTRODUCTION

Mobile augmented reality (AR) has recently gained immense popularity with the release of several smartphone applications that blend virtual content and the real environment. For instance, e-commerce apps – including IKEA Place [36], Amazon AR View [2], and Wayfair [88] – enable prospective customers to see how products fit their home in terms of both dimensions and aesthetic properties. Moreover, educational apps allow students to visualize and interact with tri-dimensional content such as life-size animals [23] or human anatomical parts [3, 17] in the comfort of their own learning environment, be it at home or in a classroom.

AR applications demand realistic rendering of virtual content in a variety of environments, therefore, they also require an accurate description of the observed scene in three dimensions (3D) [15, 81]. For this purpose,

Authors' addresses: Yunfan Zhang, Duke University, Durham, NC, USA, yunfan.zhang@duke.edu; Tim Scargill, Duke University, Durham, NC, USA, ts352@duke.edu; Ashutosh Vaishnav, Aalto University, Espoo, Finland, ashutosh.vaishnav@aalto.fi; Gopika Premsankar, University of Helsinki, Helsinki, Finland, gopika.premsankar@helsinki.fi; Mario Di Francesco, Aalto University, Espoo, Finland, mario.di.francesco@aalto.fi; Maria Gorlatova, Duke University, Durham, NC, USA, maria.gorlatova@duke.edu.

2474-9567/2022/3-ART37
https://doi.org/10.1145/3517260

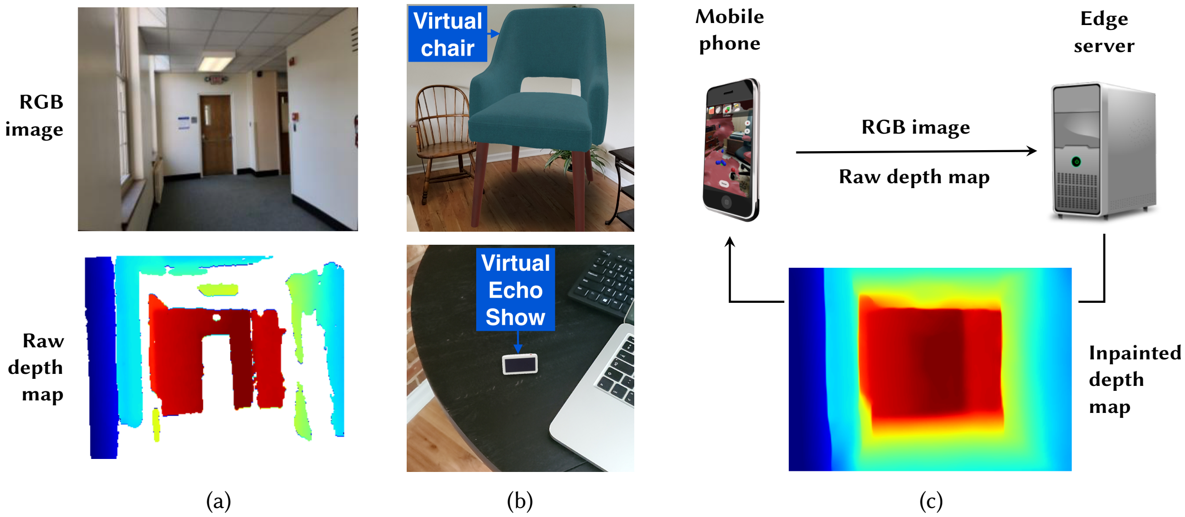


Fig. 1. (a) Representative example of environment-specific factors (top) resulting in a depth map with holes and artifacts (bottom). (b) Scale errors affecting virtual objects in Amazon AR View [2] due to inaccurate depth maps: a virtual chair rendered too large (top) and an Amazon Echo Show rendered too small (bottom). (c) InDepth leverages the edge computing paradigm to "restore" (i.e., *inpaint*) depth maps with high accuracy and a low end-to-end latency.

commonly-available devices for mobile AR – dedicated headsets and even off-the-shelf smartphones – are equipped with *depth sensors* that characterize the distance between the device and points in the real environment as a *depth map* corresponding to the pixels in the scene. Among these sensors, Time-of-Flight (ToF) cameras are particularly suitable because they provide real-time depth information with low power [7] at a high frame rate [91]. In fact, they are used in a number of consumer smartphones [15, 33, 65] and their adoption is only expected to grow in the future [90].

Unfortunately, ToF cameras suffer from several issues that affect the quality of acquired depth data. Problems in depth measurement are scene-dependent; they frequently occur in environments containing objects with extreme reflective properties, such as mirrors, polished metal surfaces, or very dark surfaces [27, 80, 98]. In these cases, extensive regions in the depth map appear as missing, resulting in large holes (Figure 1a). Inaccurate depth maps cause noticeable errors in AR applications because they determine the size (i.e., scale) of a rendered virtual object. Figure 1b shows a representative example: a virtual chair is rendered approximately 2 m in height (rather than 1 m), and the Amazon Echo Show approximately 8 cm wide (instead of 20 cm). Scale errors are particularly noticeable in AR when users directly compare virtual objects with the surrounding physical environment [81].

Several approaches have been proposed in the literature to obtain accurate depth maps (Section 2), primarily through computer vision techniques [28, 43, 92] and deep neural networks (DNNs) [32, 59]. Among them, a few key methods specifically targeted the issues caused by different types of depth sensors. Specifically, *depth completion* fills missing data in a *sparse* depth map to make it dense [48, 77]; whereas *depth inpainting* repairs possibly large regions with erroneous or missing data in a *dense* depth map [5]. The most accurate results have been achieved with DNNs, but the related computational complexity prevents them to run on mobile devices with low-enough latency to enable a smooth user experience [21, 96].

To address these issues, we design and evaluate InDepth, a real-time depth inpainting system for mobile AR based on the edge computing paradigm (Figure 1c). Accordingly, a mobile device sends both color (i.e., RGB) images and depth maps acquired with a ToF camera to an edge server for processing. The server inpaints the depth map and sends back a more accurate representation to the mobile device. In particular, InDepth leverages a novel DNN architecture and training process, grounded on an extensive evaluation of artifacts affecting depth maps obtained with off-the-shelf mobile devices. As a result, InDepth achieves high accuracy with a low latency, thereby allowing to satisfy a target frame rate of 30 frames per second despite the network overhead due to offloading. The effectiveness of InDepth is established through an extensive evaluation based on existing datasets, experiments with the implemented system, and user studies. Achieving an end-to-end latency of under 30 ms, InDepth is – to the best of our knowledge – *the only depth inpainting solution proven suitable for real-time operation in mobile AR through real experiments.* In detail, the key contributions of this work are the following.

- We created *a dataset of 18.6 thousand depth maps and their corresponding RGB images* with an off-the-shelf smartphone equipped with a ToF camera. The data – named the `ToF18K` dataset – were collected in indoor scenes typically observed in AR applications, including office spaces, laboratories, meeting rooms, and apartments. Our analysis of the data reveal that *close to 60% of the depth maps have more than 40% missing pixels* (Section 3). We systematically analyze different scene-dependent properties – namely, surface distance, orientation, brightness and reflectance – that result in holes and artifacts in the captured depth maps. Our findings are used to guide the design and evaluation of our depth inpainting system.
- We introduce a new DNN architecture with key structural properties: a *dual-encoder branch* to extract relevant features from both the RGB image and the depth map; a novel *dilated decoder block* that better estimates depth by drawing information from a larger surrounding region. Moreover, we devise a hybrid loss function and a data augmentation scheme that explicitly train the DNN to overcome the artifacts observed in our measurements and further improve performance (Section 4). These design choices allow the DNN to obtain *accurate depth maps with a latency as low as 8.7 ms, i.e., 8 times faster than the state of the art* [32] (Section 5).
- We implement InDepth on two types of edge testbeds by characterizing the latency-computation tradeoffs associated with the different phases in offloading inpainting. We also conduct a holistic evaluation of InDepth in real settings as to the factors that affect user experience when placing virtual objects, namely, their position in the view and the properties of surfaces in the physical environment. A comparison against ARCore DepthLab [15] – the state of the art in AR software for Android – highlights that *InDepth obtains a mean absolute error which is four times smaller: only 20 cm, as opposed to 78 cm in DepthLab* (Section 6).
- We finally conduct two user studies: one to characterize the impact of depth errors on previous user experiences with mobile AR, and another to evaluate the effectiveness of InDepth in rendering correctly-scaled virtual objects. The related findings highlight that scale errors were frequently observed especially by those who had exclusively used AR applications on smartphones. Moreover, *87% more participants rated the virtual objects rendered with InDepth of the correct size compared to those with DepthLab* (Section 7).

The `ToF18K` dataset and the source code of InDepth are available at https://github.com/InDepthOpensource.

2 RELATED WORK

Depth Sensing in Mobile AR. As an alternative to dedicated depth cameras (Section 3.1), several works have addressed *depth estimation* on smartphones and AR devices. Depth estimation involves obtaining a depth map through single or dual RGB cameras [26, 47, 52, 81]. These approaches require an initialization step, wherein the user moves the smartphone to derive parameters necessary for depth estimation from the captured RGB images [47, 81]. Another category of solutions relies on 3D reconstruction [56, 70] using a single camera on a smartphone. These approaches require continuous use of the mobile GPU to obtain depth maps at high frame rates, and incur significant memory overhead to build volumetric representations [81]. In contrast, ToF cameras capture depth maps without a time-consuming initialization step and at high frame rates. For this reason, we focus on improving the quality of depth maps obtained from such cameras.

Improving the Quality of Depth Maps. Many solutions have been proposed to generate complete depth maps from *sparse* depth maps obtained from LiDAR sensors [12, 28, 31, 48, 59, 77]. These works generate dense depth maps from an input containing accurate values for only 200-500 pixels *uniformly* located in the image. Another body of work focuses on refining depth maps through *depth super-resolution* or *upsampling*. Their goal is to improve the spatial resolution of input depth maps by using high-resolution color images [35, 50, 75] or the input depth map alone [63, 87]. Finally, a few algorithms based on bilateral filtering have been proposed to *denoise*, smooth, and complete sparse depth maps [16, 92]. All the approaches described above focus on sparse depth maps. In contrast, we focus on *inpainting* depth maps, wherein the input is dense but is missing values in arbitrarily-shaped and potentially large regions (as shown at the bottom of Figure 1a).

Depth Inpainting. Many solutions have been devised for depth inpainting [5, 28, 43], more recently through DNNs [32, 96]. Zhang and Funkhouser [96] propose a DNN that predicts surface normals and occlusion boundaries from an RGB image, and then use an optimization problem to obtain a complete depth map. Huang et al. [32] replace such an optimization with a secondary DNN to obtain the depth map from intermediate output, resulting in higher accuracy. In contrast to these solutions, we propose a single end-to-end encoder-decoder network to directly inpaint depth maps from RGB-D images, thereby reducing latency by eliminating the computational overhead of the optimization and of the secondary DNN. Moreover, our DNN uses novel architectural components (Section 4) that enable the network to learn depth values for missing pixels with a higher accuracy than [32, 96].

Edge Computing for Mobile AR. Given the resource demands associated with environment mapping, pose tracking and rendering virtual objects in AR, it is often desirable to offload computation to an edge server [6, 44–46, 61]. Recent works have demonstrated edge-based architectures for several mobile AR tasks, including simultaneous localization and mapping [6] and object recognition [44–46]. Edge servers are also envisioned to support the provision of AR content at low latency [71]. However, to the best of our knowledge, our work is the first to leverage an edge architecture for depth inpainting.

User Experience in Mobile AR. Results of an online survey on experiences with commercial AR were reported in [55], though this was in 2011, and the capabilities of AR have changed dramatically since then. Others have used surveys to gauge the interest and preferences of industry specialists in specific AR use cases, including AR-based tools for 3D media professionals [41], an AR-aided surgery system [58], and industrial AR user interfaces [21]. We use a similar methodology to understand users' past experiences with mobile AR. Recent work has shown how inaccurate depth maps result in unrealistic placement, occlusions [81], and lighting of virtual objects [97] in the real environment. Other research has examined incorrect positioning of virtual objects that occur due to inaccurate device pose estimation [62, 83], which may manifest as scale errors depending on the direction of displacement. However, to the best of our knowledge, we are the first to study the perception of virtual object scale errors in consumer applications resulting from incorrect depth maps.

3 DEPTH SENSING FOR MOBILE AR

This section introduces the principles of depth cameras based on Time-of-Flight, due to their prevalence in consumer AR devices [15]. It then characterizes the limitations in the depth maps acquired by such cameras due to different scene-dependent properties through an extensive dataset we collected.

3.1 A Primer on ToF Cameras in AR Devices

A Time-of-Flight (ToF) camera works by illuminating the observed scene with infrared light and measuring properties of the reflected light to estimate the distance to the points in the scene [30, 98]. In its simplest form, a ToF camera has an infrared light emitter and a receiver to detect the reflected signal (Figure 2). The camera produces a *depth map*, i.e., an image of the observed scene that encodes the distance to each point therein. There are two distinct classes of ToF cameras depending on whether emitted light consists of *timed pulses* or *continuous waves* [98]. The former directly calculates the time between sending a light pulse and receiving the reflected

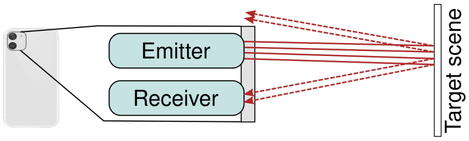


Fig. 2. Operating principles of indirect Time-of-Flight depth cameras. Light reflected from the target scene is received by a sensor; the phase shift between the emitted and the reflected signal is measured to calculate the distance.

light; whereas the latter determines the *phase shift* between modulated (as sine or square waveforms [99]) and reflected light to calculate distance. For this reason, a ToF camera employing modulated light is also known as an *indirect* ToF camera. We focus on this type as it is used in off-the-shelf devices such as Samsung Galaxy Note 10+ [65], Huawei P40 Pro [33], Magic Leap [15], and Microsoft HoloLens [34].

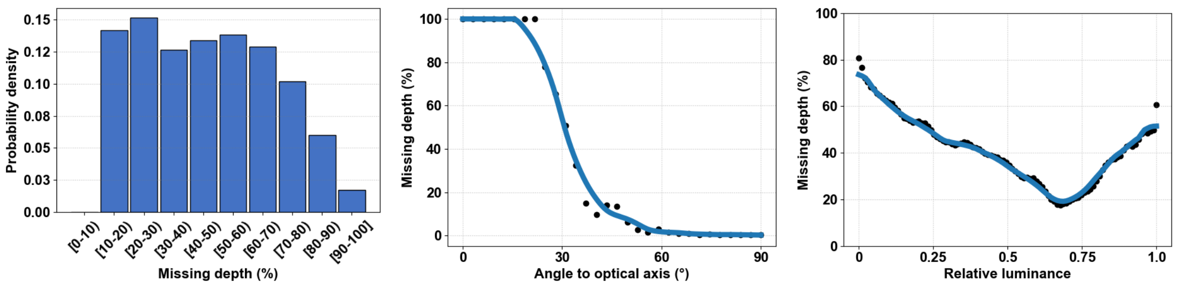
To detect phase shifts, indirect ToF cameras require multiple samples (typically four) per measured distance; the samples are obtained by modulating the emitted light at different phase steps [79]. The four samples are used to calculate the phase angle and the distance to each point in the scene. The low complexity of such calculations implies low processing power requirements [7], and, thus, *ToF cameras are expected to become ubiquitously available on smartphones* [15, 81]. In fact, ToF cameras on smartphones have been recently used in novel applications such as translating sign language [57], recognizing hand gestures [86], and improving lighting estimation for AR [97]. ToF cameras are known to suffer from several systematic errors caused by the internal temperature of the camera, integration time settings, and infrared demodulation errors [18, 19]. These errors are typically compensated by the algorithms used within the camera to calculate distances. However, errors are also produced by properties of objects in the observed scene [98], which we describe and quantify next.

3.2 Limitations of ToF Cameras

Depth maps acquired by ToF cameras suffer from measurement errors when the observed scene contains distant [22, 34, 74, 80] or off-centered surfaces [27, 80], strongly specular objects (e.g., polished metal surfaces and mirrors) [80], materials with low reflectivity [27, 80, 98], and reflections from multiple objects that interfere with each other [30]. Under these different challenging conditions, whole regions in the obtained depth map may be marked as missing, resulting in depth maps with holes. To quantify the limitations of ToF cameras, we collected 18.6K depth maps (and their corresponding RGB images) with a Samsung Galaxy Note 10+ smartphone in indoor scenes typically observed in AR applications, including office spaces, meeting rooms, and apartments. The collected dataset, called `ToF18K`, is available at https://github.com/InDepthOpensource.

Missing Depth Pixels. As a preliminary step, we evaluated the occurrence of missing pixels in `ToF18K`. Our analysis showed that 47.2% of the total depth pixels were missing in the recorded depth maps. The distribution of missing pixels in the depth maps (Figure 3a) suggests that *58% of the captured maps had more than 40% missing pixels*. For 31% of the images, more than 60% of the depth pixels were missing. The high ratio of missing pixels demonstrates that ToF cameras have several limitations despite corrections for systematic errors. Figure 4 shows images representative of common issues we observed in the captured depth maps. We explicitly focus on these next.

Impact of Distance. ToF cameras have an operating range between 3.5 m to 5 m [22, 34, 74]. The range depends on the modulation frequency of the emitted light and can be extended by reducing such a frequency at the cost of a lower accuracy [79]. The specifications for the ToF camera on the Samsung Galaxy Note 10+ are not publicly available, therefore we empirically determined the range to be 5 m through controlled experiments. Such a limited range is a source of issues in AR applications, since the distance between physical objects and an AR device can easily exceed it even in indoor scenarios. For instance, Figure 4a shows that the distant wall is not recognized in the depth map due to the camera's limited range.



(a)

(b)

(c)

Fig. 3. (a) Distribution of missing depth pixels in ToF18K. Percentage of missing depth as a function of (b) the angle between the camera's optical axis and target surface, and (c) the relative luminance of the corresponding RGB pixel in ToF18K.

Impact of Surface Orientation. A surface parallel to the camera's optical axis is generally not reliably measured by a ToF camera. This is because the infrared beam reflected from the surface at high incident angles is not strong enough to be detected [27, 80]. To characterize this issue, we carried out experiments by using 16 ArUco markers [20] arranged as a 4 × 4 grid on a smooth white wall located 2.5 m away from the smartphone. The markers enabled us to derive 3D coordinates of points on the surface, which were used to calculate the relative position and orientation between the wall and the camera. Figure 3b shows the percentage of missing depth as a function of the angle between the optical axis of the camera and the target surface. Here, 0° means the optical axis of the camera is parallel to the target surface, while 90° denotes that the optical axis is perpendicular to the target surface. We notice that missing depth pixels start to appear once the angle between the camera and target surface drops below about 60°; close to 50% of the depth pixels are missing at about 30°. The depth map entirely consists of missing pixels once the angle between the camera and target surface is 20° or below. In ToF18K, we found that the depth measurements for the floors and ceilings started to appear incomplete at distances farther than 2 m (Figure 4a) and valid depth measurements were barely available after 4 m, which supports our findings in the controlled experiment. This implies that the depth maps exhibit holes when capturing scenes with floors, ceilings, and tabletops, as these surfaces are nearly parallel to the optical axis of the camera.

Impact of Surface Brightness. ToF cameras often fail to acquire measurements on surfaces that are overly bright or dark [27, 80, 98]. Bright surfaces reflect infrared emitted from ambient sources (such as sunlight and artificial lighting) which then interfere with the active infrared beam emitted by the ToF camera. Dark surfaces absorb too much of the infrared beam emitted by the ToF camera, resulting in holes in the depth map [98]. To quantify these, we calculated the ratio of missing depth pixels in ToF18K with respect to the brightness of corresponding pixels in the color (RGB) image. We converted the RGB pixel values to relative luminance according to [37]. As Figure 3c shows, a relative luminance of 0.66 is associated with the lowest amount of missing depth data – only 18% of total pixels. The amount of missing depth data increases for other relative luminance levels, reaching 72% for the value of zero (almost black surfaces), and 50% for the value of one (very bright surfaces). Figure 4b shows an example where the ToF camera fails to measure the distance to the sofa because it is dark, whereas it is able to estimate the depth of the pillows, which have a similar distance as the sofa but are brighter.

Flares and Other Artifacts. ToF cameras suffer from additional artifacts and data loss due to undesired reflections from other surfaces in the scene [30], or from the glass covering the emitter and the receiver [53]. Such issues occur intermittently, depending on the objects in the scene and lighting conditions. As a result, the depth readings in the affected regions appear as extremely small or large, leading to artifacts in the depth map, as shown by the flare highlighted within a rectangle in Figure 4c. We observed that these artifacts often occur at the peripheral of

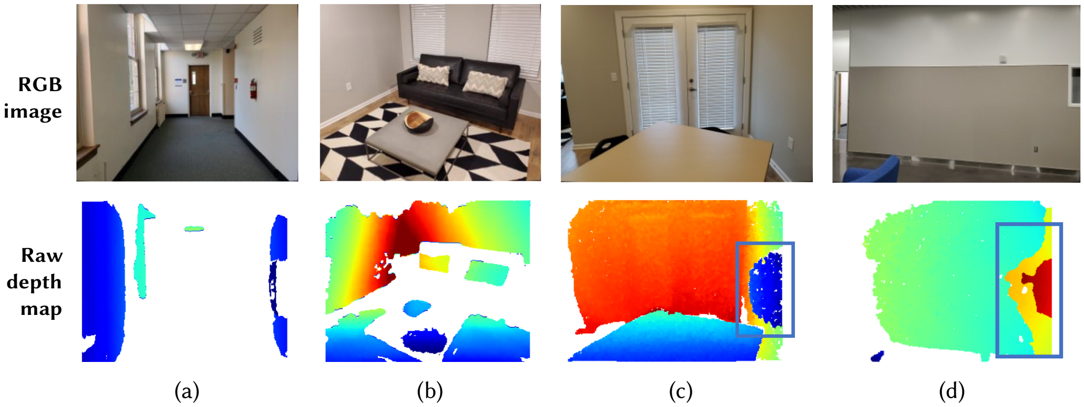


Fig. 4. Examples of input depth maps with holes: the raw depth maps contain missing data when surfaces are (a) too far or (b) dark. Depth artifacts, annotated with black rectangles in (c) and (d), caused by undesired reflections.

the depth maps, but with an irregular shape (Figure 4d). Such issues impact user experience in AR applications as they limit the user's ability to interact with parts of the observed scene.

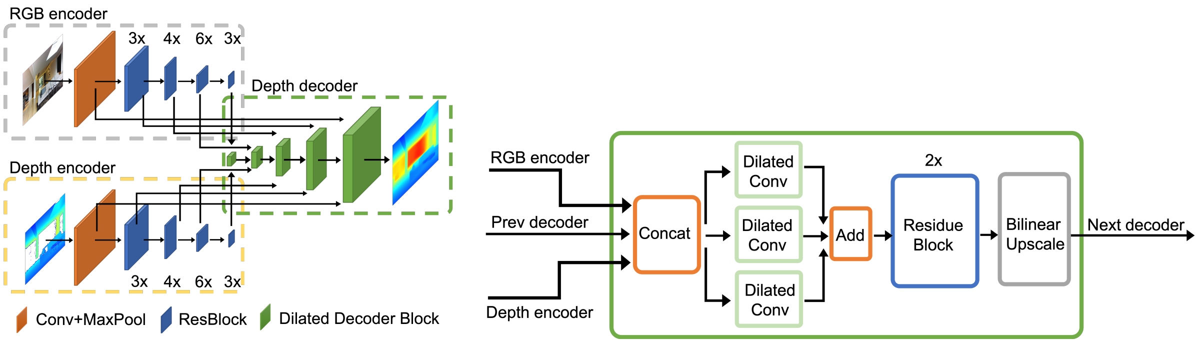
4 DNN DESIGN FOR DEPTH INPAINTING

This section presents the DNN used by InDepth for depth inpainting in terms of the related architecture, the used loss function, and the methodology employed for training.

4.1 DNN Architecture

We propose a novel DNN based on an encoder-decoder architecture with skip connections [64] between each corresponding scale of the decoder and the RGB and depth encoders to facilitate backward gradient flow from the decoder to the encoders (Figure 5a). The encoder stage employs two independent branches to extract features from the RGB and the depth inputs separately. Such a two-branch encoder architecture is specifically leveraged to enable more effective transfer learning. In fact, it allows to initialize the RGB branch of the encoder with weights pre-trained on ImageNet [14] – a large image classification dataset – and freeze the RGB encoder during the initial training to preserve the pre-trained RGB encoder weights. The encoders extract feature maps from both the RGB image and the depth map at scales of 1/2, 1/4, 1/8, 1/16, and 1/32 of the original input size. ResNet-34 [29] was employed as the backbone architecture in the encoders for two main reasons. First, the residue connections therein help mitigate the vanishing gradient problem [29, 84]. Second, ResNet-34 has a lower computational overhead than more complex DNNs such as VGG [73], which is necessary for real-time inference.

We also propose a new DNN module called *Dilated Decoder Block* for the depth decoder (Figure 5b). Several dilated decoder blocks fuse the two separate feature maps from the RGB encoder and the depth encoder, eventually restoring the internal feature maps of the DNN into an inpainted depth map. Due to the limited perceptive window, the DNN as such would tend to disregard large-scale features such as the overall geometry of the room, thereby generating inpainted depth maps that are inconsistent with these features. Consequently, multiple levels of dilated convolution [94] are employed to enlarge the perceptive window of each neuron, similar to image segmentation networks that require large perceptive fields [11, 94]. Specifically, dilated convolutions allow the decoder to better estimate missing depth by drawing information from a larger surrounding region. At each decoder block, the feature maps generated by the previous decoder block are concatenated with the feature maps extracted from the RGB and depth encoders. Then, the resulting feature map is processed with dilated convolutions with a 3 by 3 kernel size and dilation ratios of 1, 3, and 6; the outputs from the dilated convolutions



(a) (b)

Fig. 5. DNN used by InDepth: (a) architecture and (b) design of the Dilated Decoder Block. The latter leverages three channels of dilated convolution (with dilation ratio of 1, 3, and 6) to increase the perceptive field of the neurons therein.

are then added together before further processing the depth maps with two consecutive residue blocks. Finally, bilinear upsampling is leveraged to increase the width and the height of the feature maps for the subsequent level.

4.2 Loss Function

The DNN should learn the relative geometry of objects in the scene, generate an inpainted depth map with correct object contours, and prevent extreme depth inpainting errors. For this purpose, we designed a custom *hybrid* loss function as a weighted sum of Virtual Normal Loss (VNL) [93], gradients of depth estimation error, and BerHu loss [42]. Accordingly, our hybrid loss function is defined as $L(x, y) = \sum_i \lambda_i L_i(x, y), i \in \{1, 2, 3\}$, where i is the index of the individual loss function, x the predicted inpainted depth, and y the corresponding ground truth. The individual loss functions are described next.

VNL was selected due to its ability to enforce correct geometry in the scene; it was preferred to other geometry-based loss functions such as the cosine error of local surface normals as it revealed to be more robust to the noise in the ground truth for depth maps. The VNL is obtained by first taking a random selection of N pixel coordinates $\mathcal{S} = \{\{\{u_1^j, v_1^j\}, \{u_2^j, v_2^j\}, \{u_3^j, v_3^j\}\}, j = 0...N\}$, similar to [93]. Two sets of triplets of depth pixels are selected: one from the predicted depth map x and another set from the ground truth y, according to coordinates in S. The selected depth pixels are then projected into 3D points through the pinhole camera model. The resulting triplet of 3D points for a given depth map is indicated as $\mathcal{T} = \{\{P_1^j, P_2^j, P_3^j\}, j = 0...N\}$. Finally, the normal vectors are calculated for each triangle and the L1 error between the two groups of vectors is derived as the VNL:

$$L_1(x, y) = \frac{1}{N}\left(\sum_{j=0}^{N} \|\boldsymbol{n}^j(x) - \boldsymbol{n}^j(y)\|_1\right) \quad \text{with} \quad \boldsymbol{n}^j(z) = \frac{\overrightarrow{P_1^j P_2^j} \times \overrightarrow{P_1^j P_3^j}}{\left\|\overrightarrow{P_1^j P_2^j} \times \overrightarrow{P_1^j P_3^j}\right\|}. \tag{1}$$

Gradient loss is leveraged to facilitate the reconstruction of surface contours at appropriate positions with accurate and sharp object boundaries, and smooth surfaces. The Sobel operator is applied to take the gradient of the prediction error, and then the L1 norm of that gradient is considered as the actual gradient loss:

$$L_2(x, y) = \|\nabla(x - y)\|_1 \tag{2}$$

Finally, the BerHu loss is included to directly supervise on inpainted depth by heavily penalizing pixels with a high amount of errors, therefore reducing extreme errors:

$$L_3(x,y) = \begin{cases} |x-y| & |x-y| \le c, \\ \dfrac{(x-y)^2+c^2}{2c} & |x-y| > c. \end{cases} \tag{3}$$

where c is a constant describing the point at which the loss transitions from linear to quadratic. The chosen values of the parameters in the loss functions are described next.

4.3 Data Augmentation and Training

We trained our DNN on Matterport3D [10], an RGB-D dataset that contains over 194,400 RGB-D images of 90 diverse building-scale scenes, commonly used in training DNNs for depth inpainting [32, 96] and image segmentation [72, 85]. We used the same training and testing sets as as previous works on depth inpainting [32, 96], consisting of 106K and 500 RGB-D images (respectively). We describe the data augmentation and training process next; the related code is available at `https://github.com/InDepthOpensource`.

Data Augmentation. As shown in Section 3.2, depth maps captured in real scenarios are affected by several holes and artifacts with irregular shapes and appearing at different locations. The removal of these artifacts through image processing or geometry-based heuristics is particularly challenging [96]. For this reason, we incorporate a data augmentation strategy to help InDepth recognize and ignore artifacts during depth inpainting. Specifically, we extended the training data with standard data augmentation techniques including random cropping, random horizontal flipping, and random brightness, contrast and hue shifts. Furthermore, we designed a new data augmentation technique to help our DNN adapt to limitations of consumer ToF cameras. Accordingly, we explicitly augment the training data with artifacts. We first identify potential depth artifacts caused by reflections through SLIC segmentation [1] on depth maps in ToF18K. Then, we use thresholds based on the average depth of the segment to determine whether a segment is a depth artifact caused by unwanted reflections or not. After that, we perform Canny edge detection [9] on the corresponding RGB image and check if the contour of the candidate segment for a depth artifact matches an object contour in the RGB image. If there is no match, the depth segment is deemed as not caused by objects in the scene and considered an artifact. Such a technique has allowed us to obtain about 9K depth artifacts from the samples in ToF18K. During training, we randomly add these artifacts to 50% of the training samples by superimposing the artifacts onto incomplete depth maps in the training set. We also randomly invalidate pixels in the input depth map by using masks obtained from ToF18K. More specifically, we randomly select both an RGB-D image from the Matterport3D dataset and a depth map from ToF18K at the same time during training. If a depth pixel is marked as invalid in the depth map from ToF18K, we invalidate the corresponding depth pixel in the incomplete depth map drawn from Matterport3D dataset at the same pixel location. This masking scheme helped us increase the average amount of missing depth pixels in the training set from 15% to 55%, thereby making the network more robust to inputs with a large amount of missing pixels.

Training Process. We trained our depth inpainting DNN on a cluster of machines with NVIDIA GPUs by using Pytorch 1.6 and CUDA 10.1. The RGB encoder was pre-trained on ImageNet [14] and frozen for the first 12 epochs of training. Adam was leveraged as the optimizer, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. A stepped learning rate with reset was employed as the training strategy: the initial learning rate was set to 0.0004 and decreased by 60% for every 10 epochs; the learning rate was reset to 0.0002 at epochs 30 and 60. The model was trained for 80 epochs. For the loss function, the weights were empirically set to $\lambda_1 = 1.0$, $\lambda_2 = 0.3$, and $\lambda_3 = 10.0$ because such values yielded the fastest convergence in preliminary experiments targeted at finding suitable learning parameters. The VNL used $N = 100,000$ pixel coordinates to derive 3D points and the BerHu loss used constant $c = 0.2$.

Table 1. Accuracy and latency of the InDepth DNN and solutions in the state of the art for the Matterport3D dataset.

Method	**MAE** (m)	**RMSE** (m)	**1.05**	**1.10**	**1.25**	**1.25^2**	**1.25^3**	**Latency** (ms)
Bilateral [92]	0.774	1.978	0.385	0.497	0.613	0.689	0.730	1457.1
MRF [28]	0.618	1.675	0.506	0.556	0.651	0.780	0.856	685.0
AD [43]	0.610	1.653	0.503	0.560	0.663	0.792	0.861	896.0
Zhang and Funkhouser [96]	0.461	1.316	0.657	0.708	0.781	0.851	0.888	4036.6
Huang et al. [32]	0.342	1.092	0.661	0.750	0.850	0.911	0.936	70.2
InDepth DNN	**0.294**	**1.008**	**0.685**	**0.781**	**0.876**	**0.927**	**0.950**	**8.7**

5 DATA-DRIVEN DNN EVALUATION

This section evaluates the accuracy of InDepth's DNN on existing RGB-D datasets. Specifically, it first compares its accuracy and performance against the state of the art for the Matterport3D dataset. It then presents an ablation study on the same Matterport3D dataset. Finally, the section examines the ability of the DNN to generalize without fine-tuning on a different RGB-D dataset captured with different sensor types and environments.

5.1 Comparison against State of the Art

We evaluated our DNN on the Matterport3D RGB-D dataset [10] with commonly-used metrics: the Mean Absolute Error (MAE), the Root Mean Square Error (RMSE), and the percentage of pixels within the error range $t \in \{1.05, 1.10, 1.25, 1.25^2, 1.25^3\}$, defined as $\max(x/y, y/x) < t$ [32, 96]. These metrics were derived with the set of samples and the evaluation script in [32]. The ground truth of the Matterport3D test set contains data that are mislabeled: we estimated the amount of L1 error present in the Matterport3D test set to be approximately 0.19 m by comparing the absolute error between the depth pixels that are present in the depth input against the corresponding depth pixels in the ground truth. We note that the L1 error of the considered solutions in the state of the art is always above 0.3 m, therefore making the Matterport3D dataset adequate for a comparison with the InDepth DNN. The test set was used with no data augmentation. The native resolution of the images in the Matterport3D dataset is 1024 × 768 pixels; we resized the images to 320 × 256 pixels to align with the setup in [32, 96]. All experiments were carried out on a server equipped with 8 CPU cores, 16 GB of RAM, and a NVIDIA RTX 2060 GPU. We used the implementation provided by the authors for the considered schemes; for the InDepth DNN we used TensorRT 7.2 in FP16 mode. The latency was obtained as the average value of 500 trials.

Table 1 shows the performance of the InDepth DNN in terms of both accuracy and latency with respect to the state of the art in depth inpainting, which includes the DNNs proposed by Zhang and Funkhouser [96] and by Huang et al. [32]. Similar to these works, other approaches not based on DNNs are also considered for comparison purposes: a joint bilinear filtering algorithm based on a depth inpainting algorithm [92]; a depth upsampling and inpainting method based on Markov random fields [28]; and an algorithm based on linear anisotropic diffusion [43]. The table clearly shows that the InDepth DNN obtains a higher accuracy than other solutions across all the considered metrics for the Matterport3D dataset, with a significant reduction in the inference time. In fact, our proposed DNN only takes 8.7 ms on average to execute, which is more than eight times faster than the fastest scheme in the state of the art [32]. A low latency is fundamental for solutions based on edge computing, as the time needed to exchange data between the mobile device and the server might be substantial, as later discussed in Section 6.

5.2 Design Choice Evaluation

This section evaluates the chosen approaches for the design and the training of the InDepth DNN.

Table 2. Ablation study on the Matterport3D dataset. Our final DNN design (with dual branch encoders, dilated decoder blocks, and the hybrid loss function) achieves the best accuracy with minimal increase in latency.

Method	**MAE** (m)	**RMSE** (m)	**1.05**	**1.10**	**1.25**	**1.25^2**	**1.25^3**	**Latency** (ms)
MobileNetV2, Full	0.320	1.059	0.670	0.776	0.865	0.922	0.946	5.3
EfficientNet-B0, Full	0.335	1.062	0.614	0.737	0.854	0.920	0.947	6.8
ResNet-34, No DBE	0.347	1.106	0.652	0.744	0.844	0.911	0.939	6.4
ResNet-34, No DDB	0.322	1.059	0.672	0.766	0.861	0.918	0.942	7.6
ResNet-34, No HL	0.332	1.071	0.631	0.725	0.846	0.919	0.944	8.7
ResNet-34, Full (Final)	**0.294**	**1.008**	**0.685**	**0.781**	**0.876**	**0.927**	**0.950**	**8.7**

Ablation Study. We first analyzed how different design choices affect the latency and the accuracy of the InDepth DNN by performing an ablation study on the Matterport3D dataset. Specifically, we removed each of the proposed approaches – dual branch encoders, dilated decoder blocks, and the hybrid loss function – one at the time to create different DNN variants. We then observed how the related design changes impact latency and accuracy. We also explored the use of encoder networks other than ResNet-34 to evaluate the suitability of more lightweight encoders to achieve a similar accuracy with a lower latency. We trained the DNN variants with the same procedure detailed in Section 4.3 and the same evaluation methodology described in the previous section.

Table 2 shows the performance of the different DNN variants considered in the ablation study. The obtained results show that our final design (ResNet-34, Full) outperforms other DNN design options in accuracy, with only a limited penalty in latency. Recall that the InDepth DNN employs two independent ResNet-34 encoder branches for depth and RGB inputs, dilated decoder blocks in the decoder, and the hybrid loss function during training. Using two separate encoder networks for RGB and depth improves the MAE by 5.2 cm and the RMSE by 9.8 cm compared to the variant using a single encoder network for both RGB and depth (ResNet-34, No DBE). Dilated decoder blocks improve the MAE by 3.8 cm and the RMSE by 5.1 cm with respect to the variant using plain residue blocks in the decoder (ResNet-34, No DDB). Finally, using the hybrid loss function described in Section 4.2 improves the MAE by 4.8 cm and the RMSE by 6.3 cm compared to the DNN trained without hybrid loss (ResNet-34, No HL). The results also suggest that using less-expensive encoder networks such as MobileNetV2 [66] (MobileNetV2, Full) and EfficientNet-B0 [76] (EfficientNet-B0, Full) while keeping all of the proposed approaches – dual branch encoders, dilated decoder blocks, and the hybrid loss function – reduce latency at the cost of some accuracy loss. However, such a reduction in latency is not significant. Our final design (ResNet-34, Full) allows InDepth to operate in real-time with the highest possible accuracy, as later explained.

Impact of Data Augmentation. We now characterize the impact of data augmentation to mitigate the depth artifacts through a qualitative study. Specifically, we trained the depth inpainting DNN without the data augmentation technique that superimposes artifacts to the training data (described in Section 4.3) while keeping the same design and training process used in the final InDepth DNN.

We focus our attention on four representative cases drawn from ToF18K. For these, Figure 6 shows the source RGB image, the raw depth map, and the inpainted depth with and without data augmentation. Artifacts in the raw depth map are highlighted in a rectangle. The samples show that the InDepth DNN significantly reduces these artifacts by replacing the affected regions on raw depth maps with estimated depth. In particular, InDepth completely removes small artifacts such as those in Figures 6a–6c without affecting the rest of the depth map. However, the InDepth DNN cannot completely remove large artifacts such as those in Figure 6d. The inpainted depth map still presents traces of the flares in the raw depth input, however, they are barely noticeable. In contrast, the same DNN without data augmentation not only fails to mitigate the artifacts in all cases, but also amplifies them by propagating the erroneous depth values to the surrounding regions.

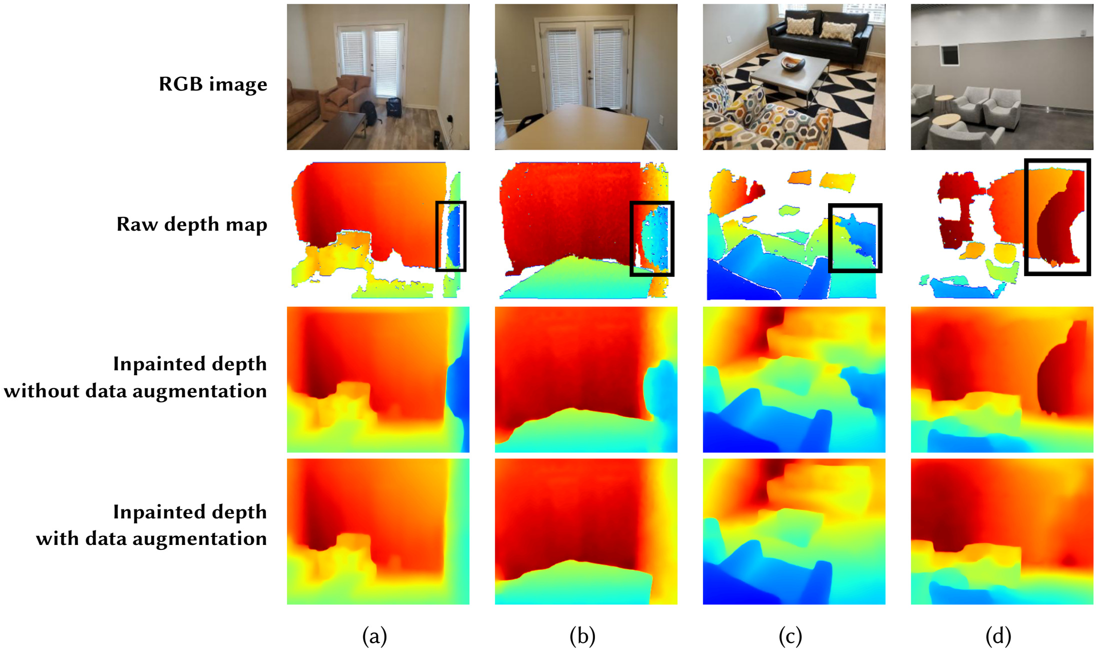

RGB image
Raw depth map
Inpainted depth without data augmentation
Inpainted depth with data augmentation
(a)
(b)
(c)
(d)

Fig. 6. Data augmentation allows the InDepth DNN to recognize artifacts and mitigate them in the inpainted depth maps, instead of simply propagating errors.

5.3 Performance with Different Types of Depth Sensors and Room Layouts

We finally evaluate the accuracy of the InDepth DNN under different types of depth sensors and room layouts so as to understand how well our solution generalizes to diverse scenarios. For this purpose, we employ the DIODE dataset [82], which was built using a laser scanner as opposed to structured light sensors used in the Matterport3D dataset [10]. The dataset contains almost 10K RGB-D images captured in indoor environments such as offices, apartments, convenience stores, and classrooms. The geometry of the rooms in the DIODE dataset is different from those of the Matterport3D dataset, and the locations of captures do not overlap. This allows us to characterize the accuracy of the InDepth DNN when inpainting depth maps acquired in room layouts different from those included in Matterport3D dataset. Accordingly, we do not perform any model training or fine-tuning on the DIODE dataset but directly apply our proposed model as trained on Matterport3D to the DIODE dataset. We use all 9,652 indoor RGB-D images in DIODE as our test set. We also augment the dataset with the same technique described in Section 4.3 to invalidate pixels and simulate incomplete depth maps from consumer ToF cameras. As a result, the average percentage of missing pixels in the input increased from 0.7% to 47%.

Table 3 details the accuracy of the InDepth DNN for the DIODE dataset. Our proposed scheme obtains 0.290 m MAE and 1.060 m RMSE on the DIODE dataset, compared to the 0.294 m MAE and 1.008 m RMSE on the Matterport3D test set. Overall, the results for the two datasets are very similar, demonstrating that the InDepth DNN is able to generalize across across different types of sensors and room layouts. It is worth noting, however, that the DIODE dataset contains a number of test samples where the content of the RGB-D images consists of mostly featureless ceilings, which are easily handled by the InDepth DNN. To isolate this effect, we also report the accuracy of the InDepth DNN on the DIODE dataset according to the elevation angles (-20°, -10 °, 0°, 10°, 20°, and 30°) of the RGB-D camera. The samples taken with 20° and 30° camera elevations mostly contain RGB-D images of the ceilings, which explains the higher accuracy compared to others.

Table 3. Accuracy of the InDepth DNN on the DIODE dataset. The DNN achieves a similar accuracy as on the Matterport3D dataset, thereby demonstrating its ability to generalize.

Dataset	**MAE** (m)	**RMSE** (m)	**1.05**	**1.10**	**1.25**	**1.25^2**	**1.25^3**
MatterPort3D	0.294	1.008	0.685	0.781	0.876	0.927	0.950
DIODE Overall	**0.296**	**1.060**	**0.773**	**0.843**	**0.909**	**0.952**	**0.970**
DIODE -20°	0.335	1.163	0.753	0.826	0.898	0.942	0.962
DIODE -10°	0.375	1.250	0.757	0.828	0.898	0.943	0.964
DIODE 0°	0.389	1.303	0.768	0.836	0.901	0.942	0.961
DIODE 10°	0.302	1.071	0.783	0.849	0.911	0.952	0.971
DIODE 20°	0.215	0.772	0.787	0.855	0.918	0.962	0.979
DIODE 30°	0.180	0.705	0.790	0.857	0.922	0.965	0.981

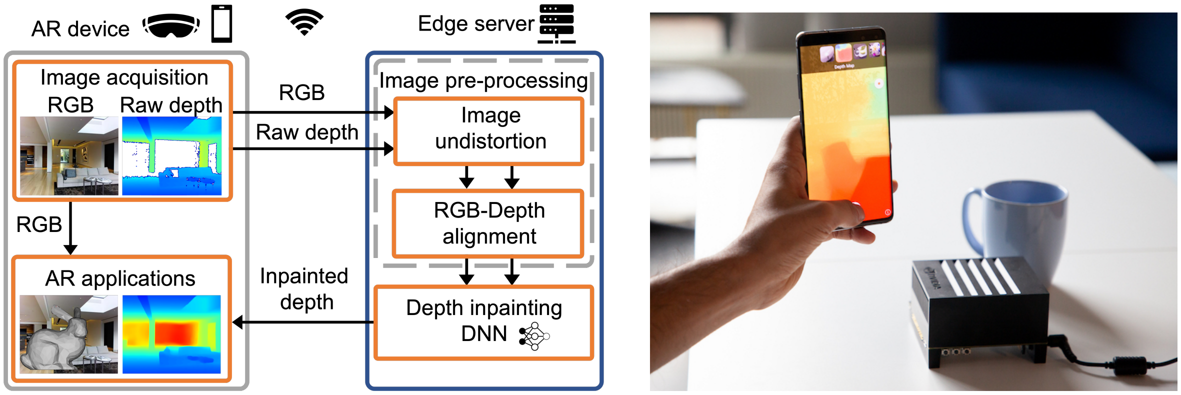
6 EXPERIMENTAL EVALUATION

This section details the software components of InDepth and the related implementation, informed by experimenting with two different classes of edge testbeds (Figure 7). It also provides an experimental evaluation of InDepth in terms of both latency and accuracy, followed by a comparison against the state of the art of software for depth estimation in mobile AR.

6.1 System Implementation

InDepth comprises three main software components (Figure 7a). An *image acquisition module* captures depth maps and the corresponding RGB images on the mobile device. Our implementation for Android devices leverages the `Camera2` API to acquire RGB images and depth maps in pairs. The RGB image is encoded in M-JPEG format, while the depth map can be either raw (i.e., in DEPTH16 format) or compressed. Furthermore, an *image pre-processing module* prepares the RGB image and the depth map for inpainting by performing two distinct operations with OpenCV and the CuPy library for GPU-accelerated computing. First, both sources are undistorted through the Brown-Conrady [8] lens distortion correction. Then, the depth map is aligned to the RGB image by projecting the depth pixels onto the RGB image according to the pinhole camera model. Finally, the *depth inpainting module* enhances the depth maps and feeds them into the inpainting DNN described in Section 4, implemented with PyTorch and optimized with TensorRT for NVIDIA hardware. The image pre-processing and the depth inpainting modules both run on the edge server.

As InDepth relies on an edge computing paradigm, all software components require sending or receiving data through the network. We investigated different options for compressing depth maps, based on the starting point that depth samples have a precision over 8 bits. Because of that, compression algorithms that support higher precision such as JPEG 2000 and PNG would appear as natural candidates. However, a preliminary evaluation has shown that the extra time required for compression exceeds the gain in transmitting a depth map with a smaller size in either case. In fact, our related experiments on 200 RGB-D image pairs revealed that JPEG 2000 takes (on average) 41 ms to encode and decode data on the mobile device, plus at least 19 ms on the edge server, resulting in a combined latency of 60 ms. Similarly, PNG takes about 12 ms to encode and decode data on the mobile device, with no less than 12 ms on the edge server, leading to a combined latency of 24 ms. As a consequence, the overall latency due to the encoding and decoding overhead is higher than transmitting an uncompressed depth map in both cases, thereby making these two formats unsuitable. For this reason, we decided to leverage JPEG, as it is less computationally expensive than JPEG 2000 and PNG – JPEG compression algorithms have also been heavily optimized and their implementations are very efficient on mobile devices as well, where encoding and decoding an image takes only a few milliseconds at most. To use such, we first apply a log mapping to the acquired depth



(a)

(b)

Fig. 7. (a) The software components of InDepth, including an image acquisition module at the mobile device as well as pre-processing and inpainting modules at the edge server. (b) One of the testbeds used to assess the implementation, consisting of an NVIDIA Jetson AGX Xavier and a Samsung Galaxy S10 5G.

map to reduce the precision of raw depth data to 8 bits, then compress the image as a JPEG. In turn, we first decompress the JPEG image and then apply the inverse of the log mapping to restore the image to the correct scale. For the log mapping of the depth data, we consider the function $y = 255 \cdot \log_{2a}(2x)$, where x is the value of the depth pixels in meters, and a is the maximum depth limit, as the log mapping function has to reduce the data to 8-bit precision. Upon decompressing, the inverse mapping $x = 2^{(y/255-1)} \cdot a^{y/255}$ is applied to restore the image. The intuition behind the proposed log mapping is that both the ToF camera and our depth inpainting DNN exhibit a higher amount of absolute error on the pixels with larger depth values. Consequently, it is beneficial to allocate higher encoding precision to the pixels with a relatively low depth, for which the ToF camera and the DNN achieve lower absolute errors.

Testbed Setup. InDepth targets achieving a smooth user experience, considered in terms of a desired frame rate of 30 FPS or – equivalently – an overall latency of at most 33 ms. Such a latency budget has guided our implementation in terms of key choices to reduce the time needed to carry out all the phases involved in inpainting depth, including the overhead due to exchanging data over the network. For this purpose, we set up two different edge testbeds, each corresponding to a representative class of edge server. The first testbed consisted of a Samsung Galaxy Note 10+ and a *workstation-class* edge server, namely, the same considered for the evaluation in the previous section – a machine with 8 CPU Cores, 16 GB RAM, and a NVIDIA RTX 2060 GPU. The second testbed, instead, included a Samsung Galaxy S10 5G and an *embedded-class* edge server – an NVIDIA Jetson AGX Xavier (Figure 7b). In both cases, the edge server was equipped with an IEEE 802.11ac WiFi router operating at the 5 GHz frequency, to which the mobile device directly connected. The two considered testbeds cover diverse scenarios in terms of the computational capabilities of the edge servers therein: one similar to a cloudlet [67], and another where edge intelligence is provided through small embedded devices with a limited power consumption [95].

6.2 Performance Evaluation

For the evaluation, we used the same software components and testbed setup described above. In addition to that, we implemented an AR application for Android by using Unity 2020. The RGB images were captured at a 480p resolution on the mobile device; the corresponding depth maps had a resolution of 320 × 240 pixels, which is commonly supported by ToF cameras on off-the-shelf AR devices [33, 51, 65]. Both the RGB and the depth

Table 4. InDepth's performance in real experiments: (a) latency and (b) accuracy.

(a)

	Latency (ms)	
Edge testbed	**W-class**	**E-class**
End-to-end (i.e., total)	26.3	36.5
Communication overhead	13.1	9.2
Image pre-processing	4.5	10.8
DNN inference	8.7	16.5
Mobile device-only	**GPU**	**CPU**
DNN inference	489.2	646.5

(b)

Metrics	**Experiments**	**Matterport3D**
MAE (m)	0.238	0.294
RMSE (m)	0.468	1.008
1.05	0.621	0.685
1.10	0.873	0.781
1.25	0.941	0.876
1.25^2	0.980	0.927
1.25^3	1.000	0.950

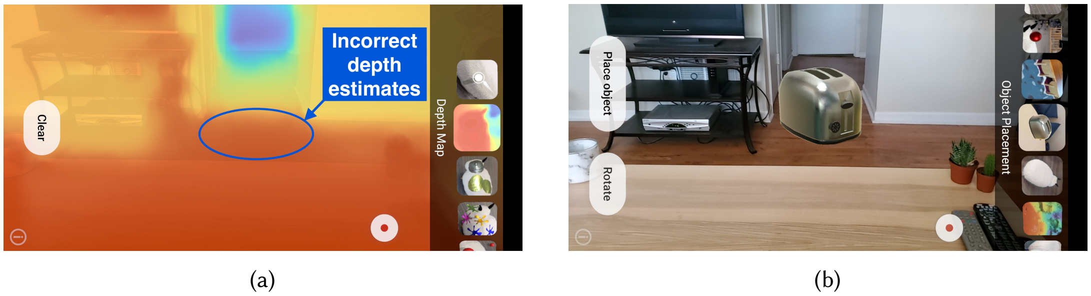
images were resized to 320 × 256 pixels to generate an inpainted depth map of the same size, as the InDepth DNN only supports inputs with dimensions that are multiples of 32.

End-to-end Latency. We first focus on the *end-to-end* latency, namely, the total latency resulting from offloading inpainting to the edge server – also including the overhead for exchanging data over the network. Specifically, such a latency includes the time elapsed between capturing an RGB-D image at the mobile device and obtaining the inpainted depth map as the actual input for the AR application. We considered both testbeds: the one with a workstation-class edge server (indicated as W-class) and that with an embedded-class edge server (indicated as E-class). We just employed raw (i.e., uncompressed) depth maps for the W-class testbed, whereas we applied the JPEG compression scheme with log mapping (described in the previous section) for the E-class testbed. In addition to our implementation, we also considered running the DNN locally at the mobile device on either the CPU or the GPU for comparison purposes.

Table 4a shows the average latency obtained over 1,000 replications of individual experiments. The table also includes a detailed breakdown of the latency according to the different phases[1] in offloading inpainting. The results highlight that *the end-to-end latency of InDepth is one order of magnitude smaller than even the fastest option for running the DNN at the mobile device.* Indeed, local execution on the mobile CPU and GPU incur a latency of 646.5 ms and 489.2 ms (respectively). Such values are not at all suitable for real-time applications. In contrast, the end-to-end latency of InDepth is below 37 ms on average for the two classes of the edge servers in the testbed. In particular, a W-class server results in an average end-to-end latency of 26.3 ms, which is below the 33 ms required to achieve the target frame rate, which demonstrates the suitability of InDepth for a smooth AR experience in real-time. The E-class server has significantly less computational resources and cannot reach the desired frame rate despite the optimizations. However, it still allows to achieve more than 25 FPS, which is commendable given its resource constraints. For both testbeds, the actual values of the latency for image pre-processing and DNN inference are rather consistent. The communication overhead, instead, exhibits a larger variability as it depends on the conditions of the wireless channel. Still, the related latency for the W-class server has a standard deviation of only 2.8 ms, and a 99th percentile of 21.0 ms, implying that the target frame rate was achieved in the vast majority of the experiments.

It is worth noting that the adopted depth map compression scheme for the E-class server is lossy, therefore, it affects the accuracy of inpainted depth map. For this reason, we carried out additional experiments to characterize the impact of depth compression on accuracy. Specifically, we re-trained the DNN by using JPEG-compressed depth maps in the same settings as in Section 4. Our results show that the accuracy is only slightly lower than

[1] For a fair comparison, the time needed to encode and decode compressed depth maps is included under communication overhead, as the primary goal of compression here is to reduce network latency.



(a)

(b)

Fig. 8. Depth map and virtual object placement in Google ARCore DepthLab: (a) the depth of the circled area on the dark brown wooden floor is underestimated; (b) a silver virtual toaster placed there appears two times bigger than it should be.

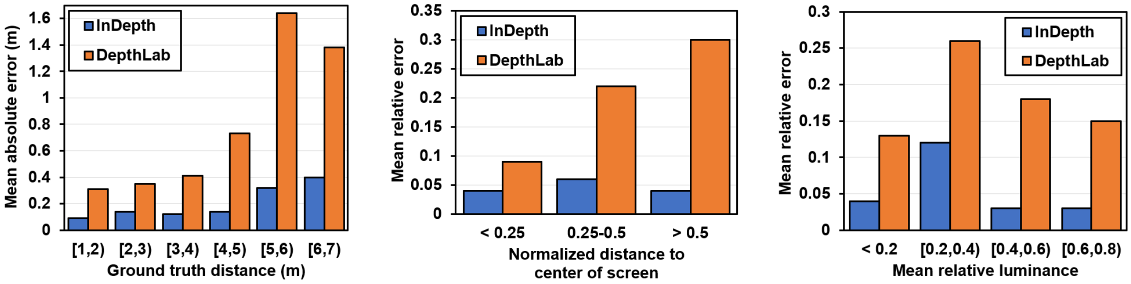
without depth compression, with a MAE of 34.7 cm, a RMSE of 1.0491 m, and a percentage of pixels within the error range between 0.613 (for $t = 1.05$) and 0.943 (for $t = 1.25^3$). These values are still almost the same as the best solution in the state of the art (Table 1). For clarity, however, we consider a W-class edge server for the rest of the experimental evaluation, as this better allows to compare with the results in Section 5.

Depth Inpainting Accuracy. To evaluate accuracy, we conducted additional experiments in a real environment for which we attained our ground truth distance by using a Tacklife HD60 laser distance meter. Specifically, we used the following methodology. We first captured an RGB-D image with the mobile device and then used InDepth to process and inpaint the acquired depth map. We then randomly picked a point in the scene, and used the laser measure to acquire the ground-truth distance between the selected point and the smartphone. We finally evaluated the accuracy of depth inpainting by comparing the ground truth to the same point derived from the output depth map from the DNN. In total, we collected 103 samples from five different rooms in environments such as apartments, office spaces, and university labs. Table 4b presents the obtained results in comparison with those for the Matterport3D dataset (in Section 5). The MAE and the percentage of pixels within error range t show similar depth inpainting accuracy in the two cases. This confirms the ability of the InDepth DNN to also generalize to depth data captured by consumer ToF sensors in real settings. It is important to note how the RMSE for the experiments is much lower than that for the Matterport3D dataset. This happens because of the difference between the environment in the two cases. In fact, the indoor scenarios we considered in the experiments have surfaces with distances ranging between one and 7 m. In contrast, the Matterport3D test set includes captures of large indoor and even outdoor spaces, wherein surfaces have much higher distance (i.e., up to 16 m) and lead to significant inpainting errors. This explains why InDepth obtains only a slightly higher MAE and much higher RMSE for the Matterport3D dataset, as the RMSE is more sensitive to extremely large errors.

6.3 Comparison with State of the Art

The following characterizes how InDepth compares with the state of the art for depth map generation in AR: the ARCore Depth API, along with the accompanying software library to interact with those depth maps, ARCore DepthLab [15]. After a brief overview of the software, the rest of the section compares the depth estimation accuracy of DepthLab with that of InDepth.

ARCore Depth API and DepthLab. The ARCore Depth API [24] is an application programming interface for Android to generate complete depth maps from raw data, allowing developers to access a depth estimate for any pixel in the camera image. The exact algorithm is proprietary, however, it is known to leverage a combination of smoothing and interpolation of the raw depth data to fill in missing regions. The raw depth data is obtained through (color) camera motion [81] or ToF cameras when available [24]. ARCore DepthLab [15] is an open-source software allowing users to view and interact with the depth maps generated by the ARCore Depth API. It supports



(a)

(b)

(c)

Fig. 9. Depth estimation accuracy with InDepth and ARCore DepthLab in terms of (a) mean absolute error for different distance ranges as well as mean relative error for (b) different screen regions and (c) surface brightness values.

visualizing depth maps as color-coded images, placing virtual objects on surfaces in the physical environment, and applying other depth-based AR effects such as occlusion and relighting. Figure 8 shows a sample depth map as shown in DepthLab as well as the corresponding RGB image. In particular, Figure 8a highlights that a depth estimate is available for all pixels; however, the edges between some objects and some finer details are poorly recognized due to smoothing and interpolation of the raw depth. As a consequence, the capture has incorrect depth estimates for a number of regions in the image. For instance, the highlighted region in Figure 8a corresponds to a dark brown floor which appears to have the same depth as the light brown table in the foreground, instead of being a greater distance from the camera (approximately twice as far).

One of the key features of DepthLab is the ability to place virtual objects on the surfaces defined by the depth map. However, these objects may be rendered with an incorrect size due to depth estimation errors: overestimates make the object appear too small, underestimates too large. Figure 8b shows a representative example of such scale errors: a virtual toaster is rendered with a size comparable to that of the TV stand next to it, which is clearly too large. This happens because the area highlighted in Figure 8a contains underestimated depth, most likely due to missing depth values for the corresponding region which are interpolated from the table in the foreground.

Depth Estimation Error. We compared depth estimation errors with InDepth and DepthLab (Figure 9). For this purpose, we ran InDepth with the experimental setup for the W-class testbed and DepthLab in a custom Unity app on the same mobile device. We conducted 63 trials in three rooms of an office building, placing virtual objects at a variety of different distances, screen touch positions and surfaces. The MAE was *0.20 m for InDepth and 0.78 m for DepthLab* across all trials, with InDepth outperforming DepthLab in 79% of trials.

Figure 9a shows the MAE obtained in the experiments as a function of the ground truth distance (measured from the smartphone camera to the virtual object placement position, using a laser meter), aggregated into ranges one-meter wide. The figure clearly shows that InDepth achieves better accuracy for all considered ranges: the MAE of DepthLab is at least 150% higher in all cases. DepthLab achieves a MAE of 0.45m while InDepth a MAE of 0.12 m for distances below 5 m, resulting in an improvement of 73%. The improvement further increases to 76% at distances over 5 m, where *DepthLab obtains a MAE of 1.51 m* whereas InDepth a MAE of 0.36 m. This is consistent with our previous observation that the quality of ToF depth measurements declines rapidly once the distance from surfaces exceeds 5 m.

We also evaluated the depth estimation error for different regions of the screen, which is important because AR users may place or view objects that are not located at the center. For this purpose, we consider the MRE (Mean Relative Error) as it accounts for the distance effects on depth estimation accuracy discussed in the previous paragraph. Figure 9b shows the MRE in the experiments for three different regions of the screen, defined in terms of how far they are from the center of the screen. InDepth outperforms DepthLab in all cases, particularly when the touch position is closer to the edge of the screen. MRE is 4% for InDepth and 9% for DepthLab when the touch position is close to the center of the screen (less than 25% of the distance from the center of the screen to the corner of the screen), but DepthLab error rises significantly when the touch position is closer to the edges of the screen (greater than 25% of the distance from the center of the screen to the corner of the screen): the MRE in those cases is 5% for InDepth and 26% for DepthLab. This is likely due to the deterioration of depth mapping quality at the edge of the imaging circle (see Section 3.2); unlike DepthLab, InDepth is able to overcome these limitations and maintain consistent depth estimation accuracy across the entire screen.

Finally, we examined the impact of surface brightness (where the virtual object is placed) on depth estimation errors. Similar to the previous case, we use the MRE as the metric of interest and derive the brightness by cropping a small square of the camera image centered on the screen touch position (approximately 20x20 cm in world space) in grayscale, then calculating the mean relative luminance within that square to measure surface brightness. Figure 9c shows the MRE for the trials within 4 different mean surface brightness ranges; InDepth improves upon DepthLab's MRE by 54–83%. For all ranges, DepthLab's MRE is greater than 10%, while InDepth's MRE is under 5% for 3 out of 4 ranges. The high relative error for both techniques in the 0.2-0.4 range (12% for InDepth, 26% for DepthLab) is due to this being the brightness level of the blue carpet in 2 of the 3 rooms we used; as we noted in Section 3.2 floors parallel to the optical axis of the camera are particularly challenging for ToF cameras, especially at distances greater than 4 m.

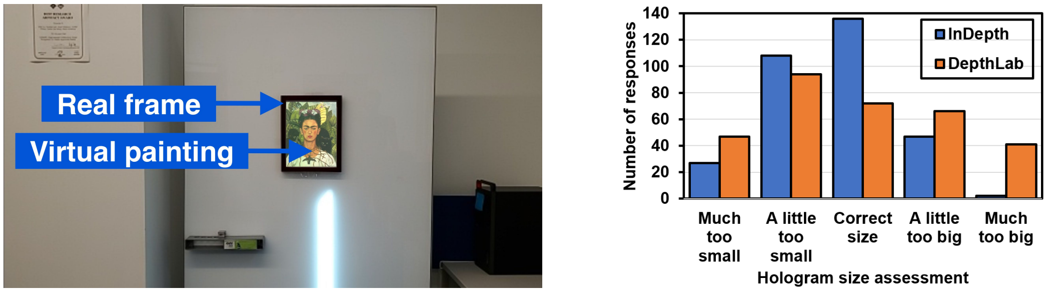
7 USER EXPERIENCE

This section presents two user studies we conducted; one to evaluate the impact of depth errors on previous experiences in AR, and one to evaluate the effectiveness of InDepth in rendering correctly-scaled virtual objects. The studies were carried out under COVID-19 restrictions on in-person experiments; as a consequence, they were both designed and executed as online surveys via Qualtrics [60]. We followed the methodology for remote user studies in [69] as well as ITU-T Recommendations P.808 [38] and P.809 [39] when designing the surveys. They were approved by the Institutional Review Board (IRB) of our university, then distributed through personal and professional networks, as well as advertised on an email list for AR and VR academics, students, and professionals. Participation in the study was voluntary and subject to informed consent.

7.1 User Experience in Mobile AR

The first study consisted of a survey on past experiences with commercial mobile AR applications, irrespective of using a specific depth sensing technology or device. We received responses from 27 individuals (52% female), who had experience ranging from less than one hour to more than one hundred hours of AR application use. 26 of the respondents had used smartphone or tablet-based AR (19 iOS, 14 Android), and 15 had used head-mounted AR such as the Microsoft HoloLens or Magic Leap One. Respondents had used a wide variety of AR applications, such as Pokemon GO [54], used by 67% of respondents, and IKEA Place [36], used by 37% of respondents. 81% of respondents had used AR applications in both indoor and outdoor settings, and 93% had used markerless AR.

When asked to consider their past experiences with commercial markerless AR, and rate how much they agreed with the statement "The generated virtual objects are always of the correct size" on a scale of 1 (completely disagree) to 10 (completely agree) *the majority of respondents indicated that virtual objects' size errors were a somewhat frequent or a very frequent issue in mobile AR* (responses: 1–3: 20.8%, 4–7: 62.5%, 8–10: 16.7%). One respondent commented "*I think markerless AR has an inaccurate understanding of the environment scale, sometimes, which causes the inaccurate overlay objects' sizes*". The issue was even more prevalent for those who had exclusively used smartphone AR (8 respondents), with *all respondents having experienced size errors somewhat frequently or very frequently* (responses: 1–3: 12.5% 4–7: 87.5%, 8–10: 0%). This reflects the fact that depth estimation, and hence virtual object rendering size, is often less accurate on smartphones than specialized AR devices like headsets.



(a) (b)

Fig. 10. Virtual object scale error user study: (a) sample screenshot from survey showing painting in frame and (b) responses aggregated by rating. Virtual paintings placed on the InDepth depth map are much more likely to be viewed as the correct size compared to those placed in ARCore DepthLab.

Moreover, 25% of respondents answered that they frequently had problems placing a virtual object on a wall, compared to 12.5% for a table. This can be explained on the basis that users are generally located farther away from walls than tables while using an AR application, and existing solutions for depth estimation often produce large errors at greater distances (see Section 6.3).

7.2 Effectiveness of InDepth Compared to State of the Art

The second study evaluated the ability of InDepth to reduce the magnitude of virtual content scale errors experienced by users – such as the incorrectly sized objects in Figure 1b or the oversized toaster in Figure 8b – compared to DepthLab, through more accurate depth maps. Specifically, participants subjectively assessed the size of placed virtual objects for the use case of online shopping apps that render copies of real objects such as IKEA Place [36] and Amazon AR View [2]. Accordingly, we consider a user shopping for a painting that they wish to place in a real frame that they already own. The reference task is represented by the user leveraging AR to place a virtual copy of a painting in the frame – to check if the painting has the correct size – as shown in Figure 10a. This was one of the tasks that respondents in our previous survey (described above in Section 7.1) told us they found challenging, placing a virtual object on a wall. It also makes it easy for users to assess if the virtual object has the expected scale, which is affected by depth estimation errors.

We were unable to allow users to rate AR content in person due to building access restrictions related to the COVID-19 pandemic, combined with the need to install our edge architecture in the same location as InDepth. Nevertheless, we replicated the scenario of a user placing a virtual painting by capturing screenshots on the AR device (a Samsung Galaxy Note 10+) upon performing this action ourselves. Specifically, these screenshots contained a virtual painting placed within a wooden real frame and were then presented to participants in online survey questions. The virtual painting should fit exactly within the wooden real frame, as shown in Figure 10b, but sometimes the painting is either too small or too big due to depth estimation error. From our trials in Section 6.3, we chose 20 locations in three different rooms (examples of these placement locations are shown in Figure 11) and generated one screenshot using InDepth and one screenshot using DepthLab. We randomized the order in which the screenshots were presented. The participants were asked to rate the size of the painting relative to the real frame on a 5-point Likert scale (*Much too small*, *A little too small*, *Correct size*, *A little too big*, and *Much too big*). We received complete responses from 16 respondents (those who rated all 40 screenshots), giving us a total of 640 responses.

The aggregated responses from the user study are shown in Figure 10b. As in our quantitative evaluation, InDepth again outperforms DepthLab, with 43% of all responses judging the virtual painting as the *Correct size*

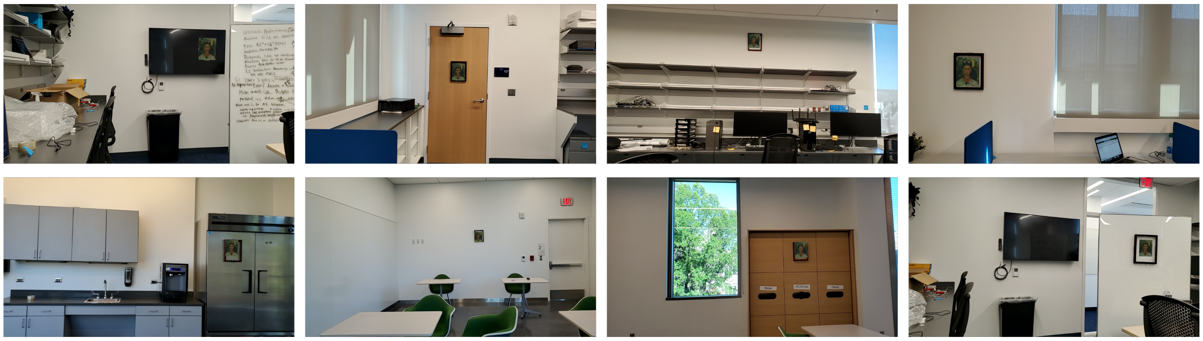


Fig. 11. Examples of the virtual painting placement locations in our user study, which included a variety of placement distances, screen positions and placement surfaces.

compared to 23% with DepthLab. Users are also much less likely to perceive large scale errors in virtual objects placed by using InDepth; only 9% responses for InDepth were in the *Much too small* and *Much too big* categories, compared to 28% for DepthLab. This is significant because small scale errors may be somewhat acceptable in a less precise task than this, but large errors are likely to degrade AR experiences.

8 DISCUSSION

In this article we developed and evaluated a real-time depth inpainting system, InDepth, for indirect ToF cameras available on Android smartphones. Recently, depth cameras based on direct ToF – marketed as LiDAR scanners [78] – have been released on a few Apple smartphones, such as the Pro versions of iPhones 12 and 13 [4]. Direct ToF cameras use timed light pulses to gather depth measurements in a scene, instead of illuminating the whole scene at once with modulated light as with indirect ToF cameras [13]. Thus, the raw depth maps obtained from direct ToF cameras are sparse [31, 48], and the task of depth completion is more suitable to completing sparse depth maps (see Section 2). Nevertheless, direct ToF cameras measure distance based on reflected light, therefore they face issues due to dark and specular surfaces [30], similar to those observed in our dataset. Updating the InDepth DNN to correct for artifacts observed with direct ToF cameras is an interesting direction for future work.

We evaluated the performance of InDepth on two types of edge devices, including an embedded-class edge server with limited computational power. To overcome the higher inpainting latency in that scenario, we compressed depth maps by using a log mapping and the lossy JPEG format, which only incurred a limited reduction in accuracy (Section 6.2). Our choice was motivated by the overhead in applying lossless compression schemes originally designed for standard images, particularly, PNG. However, other options specifically targeted to lossless compression of depth maps exist [40, 49, 89]. Accordingly, additional experiments could further explore the computation-communication tradeoffs and the use of more advanced compression schemes for depth maps.

We demonstrated that InDepth is able to reduce the virtual content scale errors that arise from inaccurate depth measurements through a user study (Section 7.2). Such a user study was limited to remote, survey-based participation due to COVID-19 restrictions; unconstrained access to the laboratory space served by our edge architecture will in future allow participants to rate AR content in person rather than through screenshots. The relaxation of these restrictions will also make it easier to install our edge architecture and conduct user studies in other locations. We will then be able to rate the performance of InDepth in an even wider range of environments, such as residential or medical buildings.

InDepth outperformed the state of the art for AR depth map generation for the scenario of a static AR device, an experiment design which enabled us to isolate depth-related virtual object scale errors specifically, rather than spatial artifacts that may occur due to inaccurate device tracking [68]. Such a static scenario is a common use case for apps such as IKEA Place [36] and Amazon AR View [2], especially given that commercial AR platforms see ToF sensors as key to “instant AR” [4], facilitating virtual object placement without prior motion. In the future, we will incorporate our depth inpainting technique into a full AR system with 6DoF tracking, and evaluate it with mobile AR users. Dynamic scenarios will also allow us to study the possible impact of motion blur in RGB images and flying-pixel artifacts in ToF depth images [13, 98]. Incorporating InDepth into ARCore is an open challenge, however: although raw depth images can be acquired in ARCore [25], the Depth API does not currently support passing processed depth images back into the AR mapping and tracking algorithm.

We focused our user study (Section 7.2) on virtual object scale errors upon object placement, the type of error most frequently observed by respondents to our AR user experience survey (Section 7.1). Other types of perceptible errors that arise from depth measurement inaccuracy could be considered in further user studies. For instance, virtual object position errors can occur as a device is moved around due to inaccurate mapping and tracking of the environment [68], and more accurate depth measurements through InDepth could help reduce the magnitude of these errors as well. Other types of depth-related errors that may be perceived by users are: incorrect occlusion effects, e.g., a virtual object appearing in front of a real object when it should appear behind it; and lighting errors, e.g., virtual shadows rendered in the wrong place due to incorrect detection of surface depth relative to a light source. These latter types of error are unlikely to occur in our scenario, in which physical space is only observable on one side of the virtual object, but fascinating avenues remain to be explored in designing user studies that do examine these factors.

9 CONCLUSION

This article presented InDepth, a real-time depth inpainting system for mobile AR based on edge computing to improve the quality of depth maps obtained with commodity ToF cameras. The InDepth DNN infers depth for missing regions in a depth map with a low latency. The DNN improves the overall quality of depth maps by relying on a data augmentation strategy that is guided by our analysis of our dataset collected from a smartphone with a ToF camera. We implemented InDepth and evaluated its performance based on different datasets and a custom AR application in real settings. Our results demonstrated that InDepth achieves high accuracy, while reducing the inference latency to as low as 8.7 ms on existing datasets. In real experiments, InDepth outperformed the industry-standard solution, Google ARCore DepthLab, in terms of both mean absolute error and users’ ratings on whether the size of rendered virtual objects was correct.

ACKNOWLEDGMENTS

This work was partially supported by an IBM Faculty Award, by the US National Science Foundation under grant numbers CSR-1903136, CNS-1908051, and CAREER-2046072, and by the Academy of Finland under grants number 326346, 319710, 332307 and 338854. We would like to thank Niki Loppi of the NVIDIA AI Technology Center Finland for his help with the implementation on the NVIDIA Jetson board.

REFERENCES

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2012. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 11 (2012), 2274–2282.

[2] Amazon. 2021. Amazon AR view. https://www.amazon.com/adlp/arview.

[3] Apple. 2021. Augmented Reality - Apple. https://www.apple.com/augmented-reality/.

[4] Apple. 2022. ARKit overview. https://developer.apple.com/augmented-reality/arkit/.

[5] Jonathan T Barron and Ben Poole. 2016. The fast bilateral solver. In *ECCV*.

[6] Ali J. Ben Ali, Zakieh Sadat Hashemifar, and Karthik Dantu. 2020. Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping. In *ACM MobiSys*.

[7] Mary Branscombe. 2018. How Microsoft is making its most sensitive HoloLens depth sensor yet. https://www.zdnet.com/article/how-microsoft-is-making-its-most-sensitive-hololens-depth-sensor-yet.

[8] Duane C. Brown. 1971. Close-range camera calibration. *Photogrammetric Engineering* 37, 8 (1971), 855–866.

[9] John Canny. 1986. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8, 6 (1986), 679–698. https://doi.org/10.1109/TPAMI.1986.4767851

[10] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. 2017. Matterport3D: Learning from RGB-D Data in Indoor Environments. In *International Conference on 3D Vision (3DV)*.

[11] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. 2018. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2018), 834–848. https://doi.org/10.1109/TPAMI.2017.2699184

[12] Xinjing Cheng, Peng Wang, and Ruigang Yang. 2020. Learning depth with convolutional spatial propagation network. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 10 (2020), 2361–2379.

[13] Ilya Chugunov, Seung-Hwan Baek, Qiang Fu, Wolfgang Heidrich, and Felix Heide. 2021. Mask-ToF: Learning Microlens Masks for Flying Pixel Correction in Time-of-Flight Imaging. In *IEEE CVPR*.

[14] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE CVPR*.

[15] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, Shahram Izadi, Adarsh Kowdle, Konstantine Tsotsos, and David Kim. 2020. DepthLab: Real-Time 3D Interaction With Depth Maps for Mobile Augmented Reality. In *ACM UIST*.

[16] Ivan Eichhardt, Dmitry Chetverikov, and Zsolt Janko. 2017. Image-guided ToF depth upsampling: a survey. *Machine Vision and Applications* 28, 3-4 (2017), 267–282.

[17] Elsevier. [n. d.]. 3D4Medical - Augmented Reality. https://3d4medical.com/support/complete-anatomy/ar.

[18] Sergi Foix, Guillem Alenya, and Carme Torras. 2011. Lock-in time-of-flight (ToF) cameras: A survey. *IEEE Sensors Journal* 11, 9 (2011), 1917–1926.

[19] Peter Fürsattel, Simon Placht, Michael Balda, Christian Schaller, Hannes Hofmann, Andreas Maier, and Christian Riess. 2015. A comparative error analysis of current time-of-flight sensors. *IEEE Transactions on Computational Imaging* 2, 1 (2015), 27–41.

[20] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and R. Medina-Carnicer. 2016. Generation of fiducial marker dictionaries using Mixed Integer Linear Programming. *Pattern Recognition* 51 (2016), 481–491.

[21] Michele Gattullo, Lucilla Dammacco, Francesca Ruospo, Alessandro Evangelista, Michele Fiorentino, Jan Schmitt, and Antonio E Uva. 2020. Design preferences on Industrial Augmented Reality: a survey with potential technical writers. In *IEEE ISMAR Adjunct (2020)*.

[22] H. Gonzalez-Jorge, P. Rodríguez-Gonzálvez, J. Martínez-Sánchez, D. González-Aguilera, P. Arias, M. Gesto, and L. Díaz-Vilariño. 2015. Metrological comparison between Kinect I and Kinect II sensors. *Measurement* 70 (2015), 21–26. https://doi.org/10.1016/j.measurement.2015.03.042

[23] Google. 2021. Experience 3D & augmented reality in search. https://support.google.com/websearch/answer/9817187?co=GENIE.Platform%3DAndroid&hl=en&oco=0.

[24] Google. 2021. Introduction to Depth on Unity targeting Android. https://developers.google.com/ar/develop/unity/depth/introduction.

[25] Google. 2022. Use Raw Depth in your Android app. https://developers.google.com/ar/develop/java/depth/raw-depth.

[26] Rostam Affendi Hamzah and Haidi Ibrahim. 2016. Literature survey on stereo vision disparity map algorithms. *Journal of Sensors* 2016 (2016).

[27] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Horaud. 2012. *Time-of-flight cameras: Principles, methods and applications*. Springer.

[28] Alastair Harrison and Paul Newman. 2010. Image and Sparse Laser Fusion for Dense Scene Reconstruction. In *Field and Service Robotics*.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[30] Radu Horaud, Miles Hansard, Georgios Evangelidis, and Clément Ménier. 2016. An overview of depth cameras and range scanners based on time-of-flight technologies. *Machine vision and applications* 27, 7 (2016), 1005–1020.

[31] Mu Hu, Shuling Wang, Bin Li, Shiyu Ning, Li Fan, and Xiaojin Gong. 2021. PENet: Towards Precise and Efficient Image Guided Depth Completion. In *IEEE ICRA*.

[32] Yu-Kai Huang, Tsung-Han Wu, Yueh-Cheng Liu, and Winston H. Hsu. 2019. Indoor Depth Completion with Boundary Consistency and Self-Attention. In *IEEE ICCV Workshops*.

[33] Huawei. 2021. HUAWEI P40 Pro. https://consumer.huawei.com/en/phones/p40-pro/specs/.

[34] Patrick Hübner, Kate Clintworth, Qingyi Liu, Martin Weinmann, and Sven Wursthorn. 2020. Evaluation of HoloLens tracking and depth sensing for indoor mapping applications. *Sensors* 20, 4 (2020), 1021.

[35] Tak-Wai Hui, Chen Change Loy, and Xiaoou Tang. 2016. Depth map super-resolution by deep multi-scale guidance. In *ECCV*.

[36] IKEA. 2021. IKEA Apps. https://www.ikea.com/us/en/customer-service/mobile-apps/.

[37] International Telecommunication Union (ITU) Radiocommunication Sector. 2005. BT.470: Conventional analogue television systems. Available at https://www.itu.int/rec/R-REC-BT.470-6-199811-S/en.

[38] International Telecommunications Union (ITU) Telecommunication Standardization Sector. 2018. P.808: Subjective evaluation of speech quality with a crowdsourcing approach. Available at https://www.itu.int/rec/T-REC-P.808-201806-I/en.

[39] International Telecommunications Union (ITU) Telecommunication Standardization Sector. 2018. P.809: Subjective evaluation methods for gaming quality. Available at https://www.itu.int/rec/T-REC-P.809-201806-I/en.

[40] Hanseul Jun and Jeremy Bailenson. 2020. Temporal RVL: A Depth Stream Compression Method. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, 664–665.

[41] Max Krichenbauer, Goshiro Yamamoto, Takafumi Taketomi, Christian Sandor, and Hirokazu Kato. 2014. Towards augmented reality user interfaces in 3D media production. In *IEEE ISMAR (2014)*.

[42] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. 2016. Deeper depth prediction with fully convolutional residual networks. In *International Conference on 3D Vision (3DV)*.

[43] Junyi Liu and Xiaojin Gong. 2013. Guided Depth Enhancement via Anisotropic Diffusion. In *Pacific-Rim Conference on Multimedia*.

[44] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *ACM MobiCom*.

[45] Qiang Liu and Tao Han. 2018. DARE: Dynamic adaptive mobile augmented reality with edge computing. In *IEEE ICNP 2018*.

[46] Zida Liu, Guohao Lan, Jovan Stojkovic, Yunfan Zhang, Carlee Joe-Wong, and Maria Gorlatova. 2020. CollabAR: Edge-assisted Collaborative Image Recognition for Mobile Augmented Reality. In *ACM/IEEE IPSN*.

[47] Zifan Liu, Hongzi Zhu, Junchi Chen, Shan Chang, and Lili Qiu. 2019. HyperSight: Boosting distant 3D vision on a single dual-camera smartphone. In *ACM SenSys*.

[48] Fangchang Ma and Sertac Karaman. 2018. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *IEEE ICRA*.

[49] Sanjeev Mehrotra, Zhengyou Zhang, Qin Cai, Cha Zhang, and Philip A Chou. 2011. Low-complexity, near-lossless coding of depth maps from kinect-like depth cameras. In *2011 IEEE 13th International Workshop on Multimedia Signal Processing*. IEEE, 1–6.

[50] Nate Merrill, Patrick Geneva, and Guoquan Huang. 2021. Robust Monocular Visual-Inertial Depth Completion for Embedded Systems. In *IEEE ICRA*.

[51] Microsoft. 2021. Microsoft Mixed Reality - Manufacturing. https://www.microsoft.com/en-us/hololens/industry-manufacturing.

[52] Yue Ming, Xuyang Meng, Chunxiao Fan, and Hui Yu. 2021. Deep learning for monocular depth estimation: A review. *Neurocomputing* 438 (2021), 14–33. https://doi.org/10.1016/j.neucom.2020.12.089

[53] James Mure-Dubois and Heinz Hügli. 2007. Real-time scattering compensation for time-of-flight camera. In *IEEE ICCV*.

[54] Niantic. 2021. Catching Pokémon in AR+ mode. https://niantic.helpshift.com/a/pokemon-go/?s=accessories&f=catching-pokemon-in-ar-mode&p=web.

[55] Thomas Olsson and Markus Salo. 2011. Online user survey on current mobile augmented reality applications. In *IEEE ISMAR (2011)*.

[56] Peter Ondrúška, Pushmeet Kohli, and Shahram Izadi. 2015. Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE Transactions on Visualization and Computer Graphics* 21, 11 (2015), 1251–1258.

[57] HyeonJung Park, Youngki Lee, and JeongGil Ko. 2021. Enabling Real-time Sign Language Translation on Mobile Platforms with On-board Depth Cameras. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 2 (2021), 1–30.

[58] Antonio Pepe, Gianpaolo Francesco Trotta, Peter Mohr-Ziak, Christina Gsaxner, Jürgen Wallner, Vitoantonio Bevilacqua, and Jan Egger. 2019. A marker-less registration approach for mixed reality–aided maxillofacial surgery: a pilot evaluation. *Journal of Digital Imaging* 32, 6 (2019), 1008–1018.

[59] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. 2019. DeepLiDAR: Deep Surface Normal Guided Depth Prediction for Outdoor Scene From Sparse LiDAR Data and Single Color Image. In *IEEE CVPR*.

[60] Qualtrics. 2021. Qualtrics. https://www.qualtrics.com.

[61] Xukan Ran, Haoliang Chen, Zhenming Liu, and Jiasi Chen. 2017. Delivering Deep Learning to Mobile Devices via Offloading. In *ACM Workshop on Virtual Reality and Augmented Reality Network (VR/AR Network)*.

[62] Xukan Ran, Carter Slocum, Maria Gorlatova, and Jiasi Chen. 2019. ShareAR: Communication-efficient multi-user mobile augmented reality. In *ACM HotNets*.

[63] Gernot Riegler, Matthias Rüther, and Horst Bischof. 2016. ATGV-Net: Accurate depth super-resolution. In *ECCV*.

[64] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*.

[65] Samsung. 2020. What is ToF camera technology on Galaxy and how does it work? https://www.samsung.com/global/galaxy/what-is/tof-camera/.

[66] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE CVPR*.

[67] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing* 8, 4 (Oct.-Dec. 2009), 14–23.

[68] Tim Scargill, Jiasi Chen, and Maria Gorlatova. 2021. Here to stay: Measuring hologram stability in markerless smartphone augmented reality. *arXiv preprint arXiv:2109.14757* (2021).

[69] Steven Schmidt, Babak Naderi, Saeed Shafiee Sabet, Saman Zadtootaghaj, and Sebastian Möller. 2020. Assessing Interactive Gaming

Quality of Experience Using a Crowdsourcing Approach. In *IEEE QoMEX*.

[70] Thomas Schöps, Torsten Sattler, Christian Häne, and Marc Pollefeys. 2017. Large-scale outdoor 3D reconstruction on a mobile device. *Computer Vision and Image Understanding* 157 (2017), 151–166.

[71] Yong-Jun Seo, Joohyung Lee, Jungyeon Hwang, Dusit Niyato, Hong-Shik Park, and Jun Kyun Choi. 2021. A Novel Joint Mobile Cache and Power Management Scheme for Energy-Efficient Mobile Augmented Reality Service in Mobile Edge Computing. *IEEE Wireless Communications Letters* 10, 5 (2021), 1061–1065.

[72] Yawar Siddiqui, Julien Valentin, and Matthias NieBner. 2020. ViewAL: Active Learning With Viewpoint Entropy for Semantic Segmentation. In *IEEE CVPR*.

[73] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[74] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. 2015. SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. In *IEEE CVPR*.

[75] Xibin Song, Yuchao Dai, and Xueying Qin. 2016. Deep depth super-resolution: Learning depth super-resolution using deep convolutional neural network. In *Asian Conference on Computer Vision*.

[76] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning*.

[77] Jie Tang, Fei-Peng Tian, Wei Feng, Jian Li, and Ping Tan. 2021. Learning Guided Convolutional Network for Depth Completion. *IEEE Transactions on Image Processing* (2021).

[78] TechInsights. 2020. Sony d-ToF sensor identified in Apple's New LiDAR Camera. https://www.techinsights.com/blog/sony-d-tof-sensor-found-apples-new-lidar-camera.

[79] Texas Instruments. 2014. Time-of-Flight Camera - An Introduction (Rev. B). Technical white paper. https://www.ti.com/lit/wp/sloa190b/sloa190b.pdf.

[80] Texas Instruments. 2019. Introduction to Time-of-Flight Long Range Proximity and Distance Sensor System Design (Rev. B). https://www.ti.com/lit/pdf/sbau305.

[81] Julien Valentin, Adarsh Kowdle, Jonathan T Barron, Neal Wadhwa, Max Dzitsiuk, Michael Schoenberg, Vivek Verma, Ambrus Csaszar, Eric Turner, Ivan Dryanovski, et al. 2018. Depth from motion for smartphone AR. *ACM Transactions on Graphics (ToG)* 37, 6 (2018), 1–19.

[82] Igor Vasiljevic, Nick Kolkin, Shanyi Zhang, Ruotian Luo, Haochen Wang, Falcon Z. Dai, Andrea F. Daniele, Mohammadreza Mostajabi, Steven Basart, Matthew R. Walter, and Gregory Shakhnarovich. 2019. DIODE: A dense indoor and outdoor depth dataset. *CoRR* abs/1908.00463 (2019). http://arxiv.org/abs/1908.00463

[83] Reid Vassallo, Adam Rankin, Elvis C. S. Chen, and Terry M. Peters. 2017. Hologram stability evaluation for Microsoft HoloLens. In *SPIE Medical Imaging 2017: Image Perception, Observer Performance, and Technology Assessment*.

[84] Andreas Veit, Michael Wilber, and Serge Belongie. 2016. Residual Networks Behave like Ensembles of Relatively Shallow Networks. In *International Conference on Neural Information Processing Systems (NeurIPS)*.

[85] Fu-En Wang, Yu-Hsuan Yeh, Min Sun, Wei-Chen Chiu, and Yi-Hsuan Tsai. 2020. BiFuse: Monocular 360 Depth Estimation via Bi-Projection Fusion. In *IEEE CVPR*.

[86] Zhengjie Wang, Yushan Hou, Kangkang Jiang, Wenwen Dou, Chengming Zhang, Zehua Huang, and Yinjing Guo. 2019. Hand gesture recognition based on active ultrasonic sensing of smartphone: a survey. *IEEE Access* 7 (2019), 111897–111922.

[87] Zhihui Wang, Xinchen Ye, Baoli Sun, Jingyu Yang, Rui Xu, and Haojie Li. 2020. Depth upsampling based on deep edge-aware learning. *Pattern Recognition* 103 (2020), 107274.

[88] Wayfair. 2020. Augmented Reality with a purpose. https://www.aboutwayfair.com/augmented-reality-with-a-purpose.

[89] Andrew D Wilson. 2017. Fast lossless depth image compression. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*. 100–105.

[90] Zhiyuan Xie, Xiaomin Ouyang, and Guoliang Xing. 2021. UltraDepth: Exposing High-Resolution Texture from Depth Cameras. In *ACM SenSys*.

[91] Zhiwei Xiong, Yueyi Zhang, Feng Wu, and Wenjun Zeng. 2017. Computational depth sensing: Toward high-performance commodity depth cameras. *IEEE Signal Processing Magazine* 34, 3 (2017), 55–68.

[92] Qingxiong Yang, Ruigang Yang, James Davis, and David Nister. 2007. Spatial-Depth Super Resolution for Range Images. In *IEEE CVPR*.

[93] W. Yin, Y. Liu, C. Shen, and Y. Yan. 2019. Enforcing Geometric Constraints of Virtual Normal for Depth Prediction. In *IEEE ICCV*.

[94] Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. In *International Conference on Learning Representations*.

[95] Xiao Zeng, Biyi Fang, Haichen Shen, and Mi Zhang. 2020. Distream: Scaling Live Video Analytics with Workload-Adaptive Distributed Edge Intelligence. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (Virtual Event, Japan) *(SenSys '20)*. Association for Computing Machinery, New York, NY, USA, 409–421. https://doi.org/10.1145/3384419.3430721

[96] Yinda Zhang and Thomas Funkhouser. 2018. Deep Depth Completion of a Single RGB-D Image. In *IEEE CVPR*.

[97] Yiqin Zhao and Tian Guo. 2020. PointAR: Efficient Lighting Estimation for Mobile Augmented Reality. In *ECCV*.

[98] Michael Zollhöfer. 2019. Commodity RGB-D sensors: Data acquisition. In *RGB-D Image Analysis and Processing*. Springer, 3–13.

[99] Fernando J. Álvarez, Teodoro Aguilera, and Roberto López-Valcarce. 2017. CDMA-based acoustic local positioning system for portable devices with multipath cancellation. *Digital Signal Processing* 62 (2017), 38–51.