Received April 12, 2021, accepted June 2, 2021, date of publication June 7, 2021, date of current version June 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3086894

Camera and Lidar-Based View Generation for Augmented Remote Operation in Mining Applications

ELIJS DIMA, (Graduate Student Member, IEEE), AND MÅRTEN SJÖSTRÖM, (Senior Member, IEEE)
Department of Information Systems and Technology, Mid Sweden University, 851 70 Sundsvall, Sweden

Corresponding author: Mårten Sjöström (marten.sjostrom@miun.se)

This work was supported by the Swedish Mining Innovation under Project 2019-05162, a joint funding venture with Vinnova, Formas, and Swedish Energy Agency.

ABSTRACT Remote operation of diggers, scalers, and other tunnel-boring machines has significant benefits for worker safety in underground mining. Real-time augmentation of the presented remote views can further improve the operator effectiveness through a more complete presentation of relevant sections of the remote location. In safety-critical applications, such augmentation cannot depend on preconditioned data, nor generate plausible-looking yet inaccurate sections of the view. In this paper, we present a capture and rendering pipeline for real time view augmentation and novel view synthesis that depends only on the inbound data from lidar and camera sensors. We suggest an on-the-fly lidar filtering for reducing point oscillation at no performance cost, and a full rendering process based on lidar depth upscaling and in-view occluder removal from the presented scene. Performance assessments show that the proposed solution is feasible for real-time applications, where per-frame processing fits within the constraints set by the inbound sensor data and within framerate tolerances for enabling effective remote operation.

INDEX TERMS Augmented reality, disocclusion, Industry 4.0, lidar imaging, mining technology, real-time rendering, remote operation, view synthesis.

I. INTRODUCTION

Industry 4.0 drives the transition to (semi-)autonomous, remotely operated work even in traditionally conservative industries such as construction and mining. While productivity is always a driving factor, another important aspect is the safety and working environment of the operators of on-site heavy machinery [1], [2]. Especially in underground mining, use of remote operation centers and partial automation can significantly reduce operator risk and the operational and capital expenditure associated with the mines [2], [3]. Changing from risky on-site, in-vehicle work to remote, computer-assisted piloting can also mediate the complications of having an aging workforce and make the employment more attractive to younger generations. Incorporating remote operation and contemporary methods of telepresence via Virtual Reality (VR) and Augmented Reality (AR) is therefore relevant to the future of the mining industry.

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Wei.

The mining industry is currently transitioning to remote operation. Due to the applied nature of the mining industry, there usually is a resistance to high-level concepts, and a desire for high-feasibility solutions with a high technical readiness level. The applied state of the art for remote operation in mines is direct image (video) transmission, similar to the systems described in [4]–[6]. Fig. 1a shows an example of a remote system in a mine, where operators are shown direct video feeds from on-machinery cameras, under various connection methods (5G, wired ethernet) and video compression levels. However, going beyond the direct presentation of camera views, there is potential in using AR, view synthesis, and range sensing from Time-of-Flight (ToF) sensors such as Light Detection And Ranging (lidar) to further enhance the operator awareness of the on-site environment [7], [8]. The benefits of augmented and indirect views have been investigated in other contexts such as underwater robot operation [9], forestry [6], hazard exploration [10], and satellite repair [11], and are likely to be beneficial for mining as well.

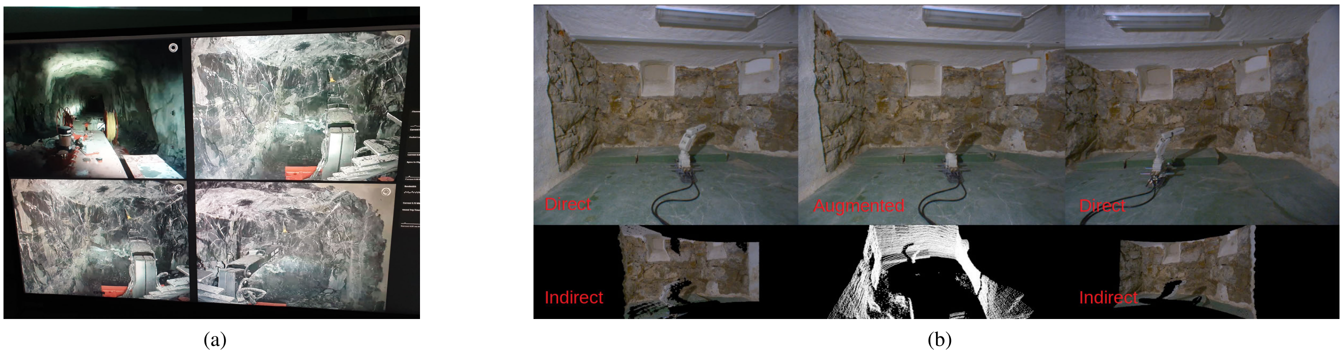

FIGURE 1. **(a) Remote operation interface for a scaling machine in a Swedish mine, showing direct rendering of on-vehicle camera views. Photo provided by Boliden AB. (b) Remote operation interface of the proposed system, showing rendering of direct, indirect, augmented and lidar views.**

There is a need to explore the use of augmented and indirect views for remote operation, in order to drive the innovation in remote operation in the mining industry context and to move the applied state-of-the-art beyond direct camera presentation. The feasibility of such a solution is, to large extent, determined by the ability to perform all data processing, fusion, and presentation within a real-time constraint set by the inbound sensor data rate. We focus on the combination of 2-Dimensional (2D) cameras and lidars because both cameras and lidars are increasingly used for control, mapping and vehicle automation in similar industries [1], [4], [12], [13]. As we see it, the main challenge lies in generating "live" augmented and indirect views of a dynamic scene, without relying on feasibility-compromising shortcuts such as pre-conditioned data (e.g. pre-scanned point clouds), off-line processing, extensive temporal delay, or green screens. Moreover, because of the mining industry's safety requirements, we cannot rely on hallucination of visually plausible image sections that do not show the actual on-site environment.

The main contribution of this work is a real-time end-to-end pipeline for capturing and rendering direct, indirect and augmented operator views of a mine-line scene from direct sensing. The system is focused on the technical and feasibility aspects of capture, processing and view generation for achieving real time performance, without ahead-of-time recordings or learned data priors. A high-level overview of the proposed pipeline is shown in Fig. 2, wherein lidar distance measurements enable the novel view generation for indirect views and in-scene occluder removal for augmented views.

The paper is outlined as follows. Section II covers the state of the art related to our work in the domains of augmented remote operation, view rendering, occlusion removal, and depth upscaling. Next, an overview of our proposed system is given in Section III, including the system constraints we adhere to. The system details are given in Sections IV, V, and VI. The system performance measurements are shown in Section VII. The outcomes are discussed in Section VIII and the overall conclusions are summarized in Section IX.

II. RELATED WORK

Augmented remote operation, or Augmented Telepresence (AT) [14], denotes applications where video-mediated communication is the enabling technology, but where additional data can be superimposed on or merged with the captured camera view as in AR. Such augmentation can be achieved on-site via using a see-through display and rendering only 3-Dimensional (3D) approximations of specific scene elements [15], or off-site via partial and full view rendering [9], [16], [17] for non-transparent displays. Unlike computer-generated imagery, this view rendering is at least partly based on the content of camera views.

There are two important problems within view rendering from multiple cameras, especially when involving sparse depth sensors such as lidar. One is occlusion removal, and the other is resolution mismatch between camera images and scene geometry, which requires some form of depth upscaling. This section gives an overview of augmented remote operation, view rendering, occlusion removal and depth upscaling works that have relevance or similarity towards our application.

A. AUGMENTED REMOTE OPERATION IN NON-ENTERTAINMENT CONTEXTS

View enhancement, or augmentation, is what distinguishes augmented remote operation from conventional VR headset based remote operation. Bejczy *et al.* [10], Yun *et al.* [5] and Omarali *et al.* [16] are examples of conventional and augmented operation, all aiming to improve the remote control of a robotic arm. In [5], [10], camera views of the scene are rendered to virtual display panels in a VR environment, without any change to the camera view content. In contrast, in [16], the camera views are fully replaced with a colored 3D point cloud model of the scene. The model is partly generated from the camera views, turning it into an extreme kind of view augmentation. This concept is taken further in [17], where the generated and presented 3D model of the scene is enhanced with virtual tracking markers overlaid in the presented 3D view.

An applied example of view-enhancing augmented remote operation is shown in [9], where Bruno *et al.* demonstrate a control interface for an underwater robotic arm. Disparity from a stereo camera is used to create the 3D structure of the scene. The 3D structure is shown in a separate view similar to [16], [17], but it is also rotated and overlaid onto a 2D

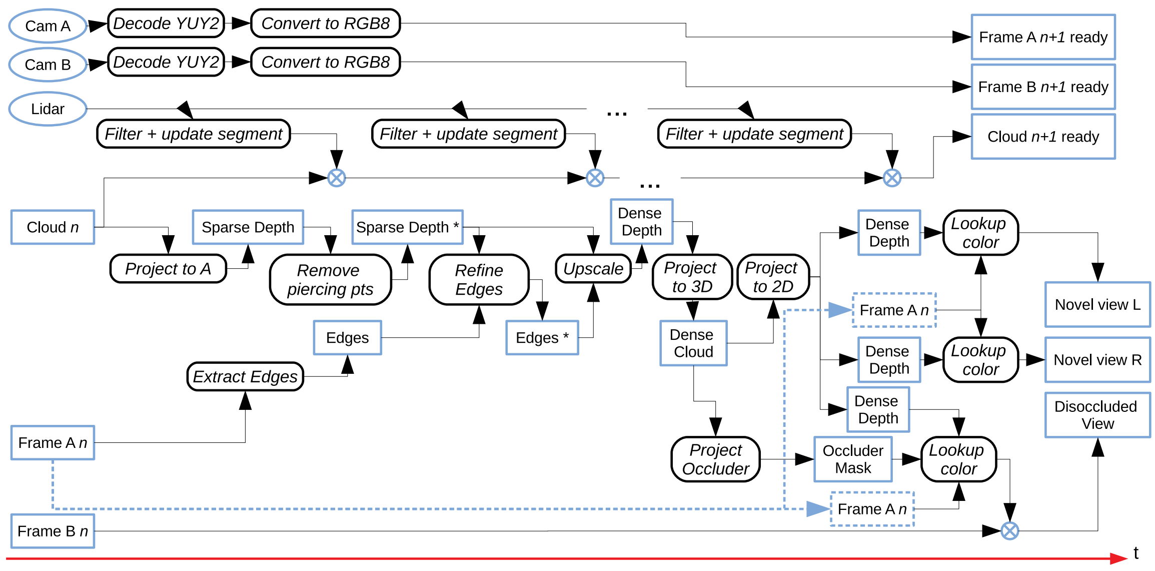

FIGURE 2. Proposed processing pipeline for rendering direct, indirect (novel), and augmented (disoccluded) views for remote operation in mining. The order of execution is depicted in sequence (left to right).

camera view, where false-color of the 3D model represents each pixel's depth value in the camera view.

Each of these examples constructs an augmented view for the operator. The augmentation relies on knowing the environment geometry, and results either in a significant alteration of parts of the presented viewpoint, or the rendering of an entirely new view inferred from the recorded data. However, in all these cases, high-resolution (i.e. densely sampled) depth is available, either from high-resolution ToF-based RGB plus Depth (RGB-D) sensors, or as disparity from rectified stereo camera pairs. Having densely sampled depth points from a high spatial resolution sensor reduces the computational effort needed to generate an augmented view. In a system such as ours, with non-rectified cameras and only sparse depth sensing by lidar, such high-resolution depth is not readily available.

B. VIEW RENDERING

View rendering for augmented telepresence is the process of compositing the scene information from cameras with additional information obtained from other sensors, presented as 2D overlays or views of the 3D model related to the 3D scene geometry. The basis for such rendering is multi-view geometry [18] and novel view rendering from RGB-D via Depth Image Based Rendering (DIBR) [19]. DIBR allows to construct a point-based 3D model of the observed scene, and project it to arbitrary virtual camera viewpoints. The 3D model can also be used to create a textured 3D mesh, or to anchor and orient the augmented rendering overlays into the presented view. More recently, free viewpoint rendering is converging with mesh-based rendering for enabling real-time applications with multiple sensors as scene data sources, as seen in [20], [21].

Rasmuson *et al.* [20] present a real-time processing pipeline for free-viewpoint rendering for videoconferencing, with focus on real time capability. They use several non-rectified 2D videocameras and projective geometry to construct a 3D mesh for a central object seen by all cameras. Upon rendering to the novel 2D view, the mesh is textured by a surface-normal dependent color blend from all source cameras. All processing is performed in CUDA and OpenGL, achieving as little as 14 ms per frame. However, in order to reach real-time processing speed, they severely limit the initial correspondence search space by using green-screen background subtraction to compute a convex visual hull, within which a 3D mesh is refined from coarse to fine through iterative correspondence search along epipolar lines. This requires both a greenscreen background, and a central object-of-interest to be covered from at least two cameras at a wide angle. Moreover, the processing is delayed by 5 frames (167 ms at 30 fps) to support a temporal filter. Transient holes in a current frame are filtered out by looking up the valid points from the surrounding 5 plus 5 frames, thus adding a fixed latency to the rendering as a tradeoff for reducing geometry flicker.

Meerits *et al.* [21] triangulate a mesh from recorded dense depth points of RGB-D sensors, instead of estimating a mesh from point correspondences like [20]. The depth from RGB-D sensors is converted to a point cloud and normals are estimated for each point. A Moving Least-Squares (MLS) method is used to jointly de-noise and rectify the clouds of two RGB-D sensors. The clouds are individually triangulated and stitched thereafter. Duplicate mesh overlaps are excluded from the stitching process, and there is no limitation on the scene content or foreground-background separation. However, with recordings from two RGB-D sensors of 640 by 480 resolution and all processing in GPU / OpenGL, the surface estimation process still takes 163 ms per frame, prior to final texturing and 2D view rendering; the majority of that time is spent on the MLS surface reconstruction stage.

C. OCCLUSION REMOVAL

We consider occlusion removal to be a specific type of view augmentation, where unnecessary scene content is removed from the operator view. The unknown set of pixels left after removing an occluder from the view is a disocclusion. Small disocclusions can be seamlessly filled with basic interpolation filters [22], [23], whereas large disocclusion removal can be addressed by image inpainting. A majority of contemporary inpainting approaches, e.g. as surveyed in [24], either rely on content-trained deep networks to synthesize a visually acceptable image section as in [25], or extrapolate missing content by inpainting replicas of the structures or texture patterns from the nearby vicinity [26].

Image inpainting methods, however, do not recover the actual revealed scene content. To remove occlusions without losing scene content, the scene must be multiply sampled from either different viewpoints or different moments in time. Different viewpoint sampling was used in e.g. [27] to reduce holes (disocclusions) within a rendered view for free-viewpoint 3D video synthesis. Due to all reference views being rectified and coplanar, [27] uses patch warping instead of full 3D reprojection. In contrast, temporal resampling for RGBD inpainting was used in e.g. [28] to achieve real-time occluder removal. The depth from a single-view RGB-D stream is used to separate distinct objects in scene, and to create a rectangular billboard plane in 3D space over the occluder. The billboard is then painted over with color from an earlier frame with the least matching error between the earlier and current scene states (minus the occluder). Temporal resampling, however, has limited viability when displaying a fully up-to-date scene state is required.

D. DEPTH UPSCALING

The relatively low resolution of depth sensors is an issue for RGB-D-based view rendering and occlusion removal, because the depth sensor does not directly measure the depth for each RGB pixel. Therefore, some processing is required to upscale the spatial resolution for the measured depth. This upscaling can be performed by tempo-spatial supersampling of the depth data, with Inertial Measurement Unit (IMU) and position-tracking [29]. However, this is only feasible for static scenes and moving sensors. For dynamic scenes, the prevalent approach is to use the corresponding RGB image as guide for depth upscaling [30]–[36].

Both [30] and [31] project the low-res depth as points onto the RGB image plane, and use the RGB image edges for global optimization penalty terms in a depth diffusion process. Zakeri *et al.* [33] extend [31] to combine multiple depth sensor readings, using both the RGB edges and stereoscopic consistency for the optimization penalty. However, such global optimization methods are difficult to parallelize [37], and tend to have prohibitive computational costs: [30] reports 318.2 ms to upscale the depth from 160 × 120 px to 810 × 610, and [31], [33] report up to 572.38 s for depth upscaling to 960 × 540 px. Plank *et al.* [32] bypass the optimization performance issue by treating upscaling as a weighted interpolation of valid depths within a nearby pixel patch. Edges from color image are used as preemptive interrupts during the marching within the pixel patch. Such upscaling can be effectively parallelized, and [32] report as little as 100 ms on a mobile GPU and 8 ms on a desktop GPU. However, [32] upscales a 288 × 256 px depth image to 640 × 480 px, so the relative resolution increase is low and interpolation kernels can be small. Furthermore, a specific RGB-D camera is used where the depth and color sensors are less than 1 cm apart, and reprojection artefacts prior to upscaling are negligible and can be ignored.

Recently, the RGB-guided depth upscaling approach has been combined with learned feature extraction and blending through e.g. a Convolutional Neural Network (CNN) [34], Residual Network (RN) [35], or a Conditional Random Field (CRF) [36]. Ni *et al.* [34] extract the edges of an RGB image, and perform bicubic upsampling on a low-resolution depth map. Both are then fed through a dual-stream convolutional network with two sets of learned feature extractors followed by a sum layer. Chen and Gao [35] also use a two-stream network, with one stream downsampling the RGB image to the depth map resolution and upscaling it back to original size, using downsampling weights as additional layers in the upsampling stage. The second stream takes the depth map and uses the other stream's upsampling weights as extra input for its own upsampling layers. However, [34], [35] are designed for single-view RGB-D scenes, where the depth map is continuous, regular, and object adjacency is preserved, unlike lidar scans with irregular, discrete point samples. Weerasekera *et al.* [36] use a single-view depth-prediction CNN to produce virtual input depth maps as parameterizations for the CRF. The method is tested on RGB-D and lidar data, however the results are dependent on training the depth predictor CNN on a high-quality dataset of feature-rich scenes, which may not be available in the context of mining. Moreover, [36] report a CRF inference time of 100 ms for lidar upscaling to 609 × 160 resolution, plus a 50 ms overhead for generating the CNN predictions.

III. SYSTEM OVERVIEW

We address the problem of generating a real-time remote operator interface with augmented and indirect views in addition to the direct camera views, in an effort to demonstrate the feasibility of such a system in the context of underground mining. The system is designed to work with live 2D cameras and a 360-degree scanning lidar, and to produce an augmented view in which an object of interest (an occluder) is removed from the view and replaced with the appropriate background. We choose to not rely on estimated texture inpainting or content generation from learned priors, because for mining applications, it is crucial for the operator to see the actual rock face with protrusions and cracks, instead of seeing a similar-looking generated texture.

Figure 2 outlines the major parts of the proposed system, which are separated into sensor data preprocessing (Sec. IV), view generation processing pipeline (Sec. V), and the final viewport rendering (Sec. VI). At a high level, the sensor preprocessing stage works concurrently with the view generation pipeline, in order to reduce the processing time between subsequent final render calls. The overall procedure is also summarized in Algorithm 1. In general, we denote a 2D image by $\boldsymbol{I}$, a 2D depth map by $\boldsymbol{D}$, a 2D edge map by $\boldsymbol{E}$, and a 3D point cloud by $\boldsymbol{P}$.

Algorithm 1 High-Level Algorithm of the System

```
1:  for every RGB camera $i$ do                                          ▷ Sec. IV-A
2:      $I^i$ ← convert frame from YUY2 to RGB8
3:  end for
4:  for every Lidar $j$ do                                               ▷ Sec. IV-B
5:      Temporal normalization of scan packet $s^j$
6:      $P^j$ ← Accumulate $s^j$ to lidar scan frame
7:  end for
8:  Set source view $i$ = 'src', target view $i$ = 'trg', source
    cloud $j$ = 'scl', novel views 'n1', 'n2'
9:  $E^{src}$ ← Extract edges from $I^{src}$                             ▷ Sec. V-A
10: $D^{src}$ ← Project $P^{scl}$ to plane of $I^{src}$                  ▷ Sec. V-B
11: $D_1^{src}$ ← Remove piercing points from $D^{src}$                  ▷ Sec. V-C
12: $E_1^{src}$ ← Refine $E^{src}$ using $D_1^{src}$                     ▷ Sec. V-D
13: $D_2^{src}$ ← Gen. dense depth from $D_1^{src}$, $E_1^{src}$         ▷ Sec. V-E
14: $P^{3d}$ ← Proj. $D_2^{src}$ to 3D, remove transients                ▷ Sec. V-F
15: $M^{trg}$ ← Project occluder from $P^{3d}$ to trg as mask
16: $D^{trg}, D^{n1}, D^{n2}$ ← Project $P^{3d}$ to trg, n1, n2
17: $I^{n1}, I^{n2}$ ← Lookup color from $I^{src}$ using $D^{n1}, D^{n2}$   ▷
    Sec. V-G
18: $I_{aug}^{trg}$ ← Lookup color from $I^{src}$ using $D^{trg}$ in $M^{trg}$ region
19: Present $I^{src}, I^{trg}, I_{aug}^{trg}, I^{n1}, I^{n2}, P^{scl}$   ▷ Sec. VI
```

A. SYSTEM CONSTRAINTS

The system is designed around two non-rectified Basler daA1600-60uc RGB cameras with mostly overlapping views approx. 1.1 m apart, 3 m away from the scene background, with an Ouster OS-1 360-degree scanning lidar near one of the cameras, in order to mimic a sensor layout that might be fitted to a mining machine (such as Jama SBU 8000E) at half scale. The Ouster lidar has a 10 Hz scan frequency, which sets a 100 ms constraint for real time generation for the augmented and indirect views. According to [5], effective remote operation is possible starting at 15 Frames per Second (FPS), which means the actual constraint is 66 ms.

The mining setting implies a system unable to reuse a pre-existing model of an already created mine tunnel, or a pre-existing map of the rock surface. Both foreground and background are to be considered potentially mobile, with unknown prior geometry and non-diffuse 5200K LED illumination pointing in the camera view direction. The scene setup is shown in Fig. 3. The lidar and camera data streams are sent either directly or via Ethernet LAN to a desktop with a Ryzen 5 3600 CPU, 16GB of RAM and an Nvidia RTX 3060Ti GPU for all processing and rendering. The output resolution of each of the generated views is equal to the camera resolution (1600 × 1200 px) for consistency between the generated views and direct-from-camera video. Each of the generated views is constructed in its own buffer to allow different view layouts; the layout shown in Fig. 1a is just one example of a potential operator view.

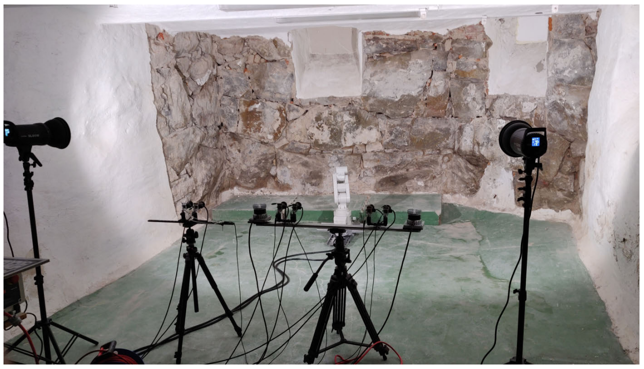

FIGURE 3. Test scene setup with cameras and lighting, with remotely actuated robotic arm as in-scene occluder.

IV. CAPTURE AND PREPROCESSING

Preprocessing is distinct from the main processing pipeline in that the preprocessing of newly arrived data occurs in parallel with the main processing of previously arrived data. Preprocessing focuses on resolving sensor-specific issues of inbound sensor data formats and contents. For the 2D cameras, this means converting the inbound raw data to the 32-bit RGB8 image format. For the lidar, this means assembling and denoising the 360-degree scan of 1024 × 64 points with $[X, Y, Z]$ coordinates.

A. THREADING AND CAMERA PREPROCESSING

Each incoming frame from each camera is either converted from 24-bit planar YUY2 pixel format to 32-bit non-planar RGB8 in case of direct connection, or decoded directly to RGB8 from an inbound h.264 stream via a GStreamer pipeline. Each camera is assigned its own handler thread that consumes the inbound frames and passes the newest RGB8 buffer onto the main thread, as shown in Fig. 2. Triple buffering is used for the frame data passover to ensure that the main render thread always has the most recent camera frame available, with neither thread needing to wait for the other. View rectification is not performed, because the direct camera views still need to be displayed for the operator, and rectification can significantly distort the viewing plane [20], making such views unsuitable for direct presentation.

B. LIDAR TEMPORAL FILTERING

Each 360-degree lidar scan (i.e. lidar frame) is assembled over 100 ms from a continuous stream of segments that are recorded as the sensor is spun, and sent over Ethernet via UDP. With spinning lidars and UDP streaming, both intermittent packet loss and lidar point drift have to be handled [38]. Furthermore, flash lidars such as the Ouster OS1 have some error in the distance estimation of each beam, resulting

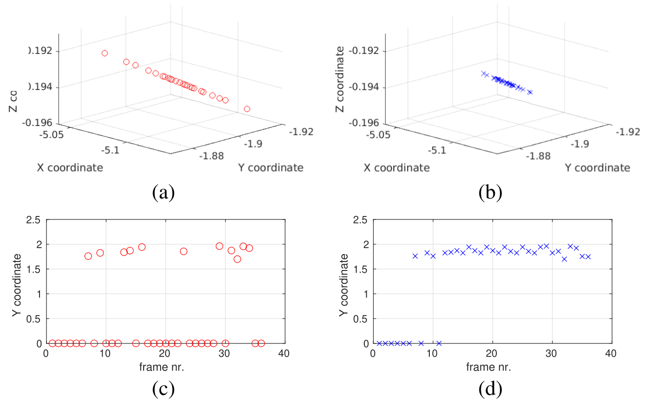

FIGURE 4. **(a) Randomly selected lidar point recording a static environment, sampled over 36 frames, showing oscillation along the lidar scan ray. (b) Same point, scan and samples as in (a), with filtering described by** (1). **(c) Another randomly selected lidar point, showing intermittent missing measurements ($Y = 0$) due to noise or packet loss. (d) Same point, scan and samples as (c), with filtering described by** (1).

in points that oscillate, i.e. shift closer/farther from frame to frame. Figure 4a shows an example of such oscillation.

Correcting such error by temporal filtering at the frame level would cause additional delay per frame (150 to 220 ms in practice due to additional frame storage and frame format conversions). Instead, we perform temporal noise filtering and missing-packet handling by applying (1) at the segment level during the frame assembly in the lidar low-level API. Figure 5 illustrates this principle.

$$\boldsymbol{p}_n = \begin{cases} \boldsymbol{p}_{n-1}, & \text{if } \boldsymbol{p}_n = \varnothing \vee \boldsymbol{p}_n = 0 \\ \boldsymbol{p}_n, & \text{if } \boldsymbol{p}_{n-1} = \varnothing \vee \boldsymbol{p}_{n-1} = 0 \\ & \vee\ |\boldsymbol{p}_n - \boldsymbol{p}_{n-1}| > \omega_{\text{diff}} \\ \boldsymbol{p}_{n-1}\omega_{\text{temp}} & \\ +\boldsymbol{p}_n(1-\omega_{\text{temp}}), & \text{otherwise} \end{cases} \tag{1}$$

where $\boldsymbol{p}$ is a 3D point $[X, Y, Z]$, n is the lidar scan iteration (i.e. "frame number"), ω_{diff} is the threshold of acceptable point oscillation, and ω_{temp} is the weight of temporal smoothing. In effect, (1) applies filtering on all points that appear to oscillate, but not on the points that may contain actual scene movement. This segment-level approach is used because at the moment of new segment arrival, all $\boldsymbol{p}_n$ and $\boldsymbol{p}_{n-1}$ for that segment are readily available and do not require additional buffering or storage of past frames. This adds less than 1 ms to the segment update, fitting seamlessly within the time gap between two successive segments.

Each full lidar frame, once accumulated, is passed to the main processing thread via triple-buffering. During this handover, the axes of the lidar scan are transformed to align the left-handed coordinate system of the lidar with the right-handed coordinate system of the camera calibration. The range scale is also converted from meters to millimeters, and the point positions are transformed to comply with the global coordinate system of all sensors.

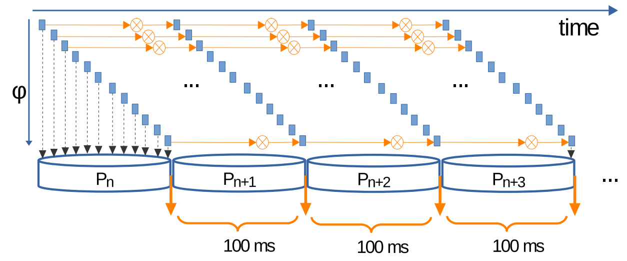

FIGURE 5. **Accumulation of lidar scan P over time from a continuous stream of small packets that cover a fraction of the scan range φ, and packet denoising across frames without delaying the frame-to-frame time interval. Number of depicted lidar packets is less than actual number of lidar packets for a 360-degree frame.**

V. PROCESSING PIPELINE

The main processing pipeline focuses on creating the indirect novel views $\boldsymbol{I}^{\text{n1}}, \boldsymbol{I}^{\text{n2}}$, and the composite disoccluded view $\boldsymbol{I}^{\text{trg}}_{\text{aug}}$. Data inputs provided from the pre-processing stage are the lidar scan $\boldsymbol{P}^{\text{scl}}$ and two camera views: the source image $\boldsymbol{I}^{\text{src}}$ and target image $\boldsymbol{I}^{\text{trg}}$. The processing can be broken into sequential steps V-A to V-G, which are called from the main render thread. As such, the execution time of these steps determines the apparent frame rate of the generated view content. Steps V-A to V-E focus on generating a dense, high-resolution per-pixel depth for $\boldsymbol{I}^{\text{src}}$ from the initial, sparsely measured lidar depth. Steps V-F and V-G are responsible for high-resolution depth projection and novel view generation.

A. EDGE EXTRACTION

Upscaling the sparse lidar depth to a dense depth map is dependent on using the structure of the Color (RGB) content as upscaling guide, similar to the upscaling methods in [30]–[36]. The first step in extracting a usable guide structure is the edge extraction from the source RGB view $\boldsymbol{I}^{\text{src}}$. A 3-pixel wide Canny filter from OpenCV [39] is used to extract an edge image $\boldsymbol{E}^{\text{src}}$ from a grayscale $\boldsymbol{I}^{\text{src}}$ blurred with a 5-pixel wide Gaussian kernel. Once extracted, $\boldsymbol{E}^{\text{src}}$ is copied to the GPU memory. The Canny filter is chosen following the example of [31], as a fast way of obtaining the first-approximation edges.

B. SPARSE POINT PROJECTION

The sparse depth measurements $\boldsymbol{P}^{\text{scl}}$ from the lidar are projected from 3D to the 2D image coordinates of $\boldsymbol{I}^{\text{src}}$, using the *'projectPoints()'* function in OpenCV. The 360-degree lidar scan $\boldsymbol{P}^{\text{scl}}$ is reduced by approx. 50% prior to the projection, by removing all points with a negative Z coordinate. This by definition affects only out-of-view points, since $\boldsymbol{I}^{\text{src}}$ serves as the origin of the global coordinate system. The projected 2D points that lie within the view frustum of $\boldsymbol{I}^{\text{src}}$ are copied to the GPU memory as a sparse depth map $\boldsymbol{D}^{\text{src}}$; from here on, all pipeline steps occur within the GPU domain.

C. PIERCE-THROUGH POINT REMOVAL

Pierce-through points are an inevitable consequence of 3D to 2D down-projection in scenarios where the 3D scan has

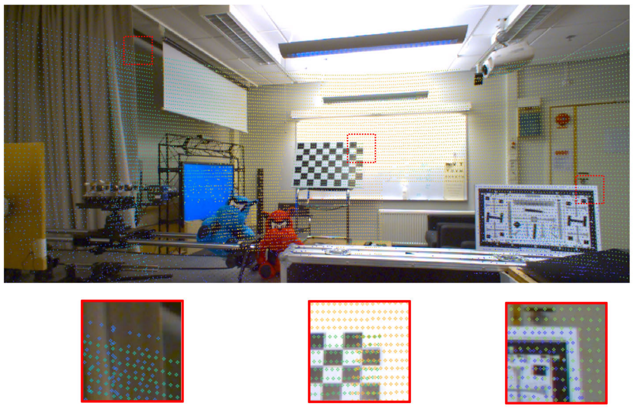

FIGURE 6. A lidar scan projected onto a camera view, exhibiting "pierce-through points" due to different positions of camera and lidar. In highlighted areas, points belonging to background are projecting between points belonging to a foreground object.

a different point of origin than the 2D projection, and particularly where the 3D measurement is sparse and irregular. We use the term "pierce-through point" to refer to a point on a more distant object projecting to a 2D position between two points belonging to a less distant object. Fig. 6 shows an example of pierce-through points.

Pierce-through points need to be detected and removed from the sparse $\boldsymbol{D}^{\text{src}}$ depth map, before depth upscaling can take place. A 16-by-16 pixel window with 4 quadrants (top-left, top-right, bottom-left, bottom-right as shown in Fig. 7a) is centered on each non-empty pixel p^{src} of $\boldsymbol{D}^{\text{src}}$, and Algorithm 2 is applied to remove pierce-through points and produce $\boldsymbol{D}_1^{\text{src}}$. In essence, we assume that if there exist nearer points in diagonal quadrants, then p^{src} maps to a pierce-through point due for removal. A depth threshold ω_{thres} is used to allow minor surface irregularities. A threshold value of 250 mm is used in this paper.

Algorithm 2 Pierce-Through Point Removal

```
1:  Given pixel $p^{\text{src}}$ and quadrants $q_1, q_2, q_3, q_4$ in $\boldsymbol{D}^{\text{src}}$:
2:  for every $q \in \{q_1, q_2, q_3, q_4\}$ do
3:      Set flag($q$): false
4:      for every pixel $p$ in $q$ do
5:          if $p \neq 0 \wedge p < p^{\text{src}} - \omega_{\text{thres}}$ then
6:              Set flag($q$): true
7:          end if
8:      end for
9:  end for
10: if $(\text{flag}(q_1) \wedge \text{flag}(q_3)) \vee (\text{flag}(q_2) \wedge \text{flag}(q_4))$ then
11:     $p^{\text{src}} \leftarrow 0$
12: end if
```

D. DEPTH EDGE DETECTION AND MASKING

The Canny filter used in step V-A cannot distinguish between a change in texture and a change in depth. We are only interested in edges which correspond to a change in depth, to use those edges as guidance for depth upscaling. We therefore mask the Canny edge map $\boldsymbol{E}^{\text{src}}$ with an estimated mask of depth edge regions, similar to [31]. Unlike [31], we have

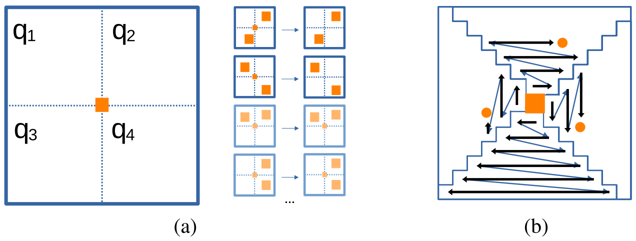

FIGURE 7. (a) Search window for finding and removing pierce-through points. The four smaller patterns illustrate the two cases that lead to point removal, and two cases that do not. (b) Search window for finding nearest depth points in top, bottom, left, right direction for the depth edge detection kernel. Arrows indicate the search pattern in the four cardinal directions.

irregular sparse depth in $\boldsymbol{D}_1^{\text{src}}$, so a map of depth discontinuities ($\boldsymbol{M}^{\text{disc}}$) must be constructed. We do so by using a custom kernel, shown in Fig. 7b, on each depth pixel p_1^{src} in $\boldsymbol{D}_1^{\text{src}}$ to find the nearest depth in each of the four cardinal directions (top, bottom, left, right). This kernel is used because $\boldsymbol{D}_1^{\text{src}}$ is irregularly populated and does not carry any information about the location of nearest adjacent points. Once a nearest-neighbor point p^{nn} is found in at least one of the four directions, a depth edge test in (2) is applied with a depth threshold $\omega_{\text{thres}} = 250$ mm.

$$\begin{aligned} &|p_1^{\text{src}} - p_{\text{top}}^{\text{nn}}| > \omega_{\text{thres}} \vee |p_1^{\text{src}} - p_{\text{bot.}}^{\text{nn}}| > \omega_{\text{thres}} \\ &\quad \vee |p_1^{\text{src}} - p_{\text{left}}^{\text{nn}}| > \omega_{\text{thres}} \vee |p_1^{\text{src}} - p_{\text{right}}^{\text{nn}}| > \omega_{\text{thres}} \\ &\implies p_1^{\text{src}} \equiv \text{edge} \end{aligned} \tag{2}$$

If $p_1^{\text{src}} \equiv$ edge, a 10 by 10 pixel window around p_1^{src} is marked in $\boldsymbol{M}^{\text{disc}}$ as a viable depth discontinuity. The masked region is intentionally left large, as precise position is expected to come from edges in $\boldsymbol{E}^{\text{src}}$. The resulting $\boldsymbol{E}_1^{\text{src}}$ is created by removing all $\boldsymbol{E}^{\text{src}}$ edges that do not lie in viable discontinuity regions of $\boldsymbol{M}^{\text{disc}}$.

E. DEPTH UPSCALING

The depth upscaling step is a diffusion of known sparse depth points in the depth map $\boldsymbol{D}_1^{\text{src}}$, using the edges in $\boldsymbol{E}_1^{\text{src}}$ as diffusion guides. We treat this depth diffusion as an interpolation problem in a manner similar to [32], in order to facilitate fast computation (as compared to global optimization based solutions). For each unknown depth p_{o} in $\boldsymbol{D}_1^{\text{src}}$, a kernel is used to select valid neighboring known points. The selection process is illustrated in Figure 8. For each neighboring known point p_n within the kernel, a narrow marching ray is cast from the position of p_n to p_{o}, testing for edges in $\boldsymbol{E}_1^{src}$ along the ray path. Any encountered edge on the path from p_n to p_{o} disqualifies p_n from the interpolation. Unlike [32], we also expand the path width during edge search to three pixels (best-fit pixel and two adjacent neighbors approximately orthogonal to path direction) to avoid missing edges due to aliasing.

The marching selection kernel produces the set $\mathcal{D}$ of the valid depths p_n for interpolation. The depth at the kernel

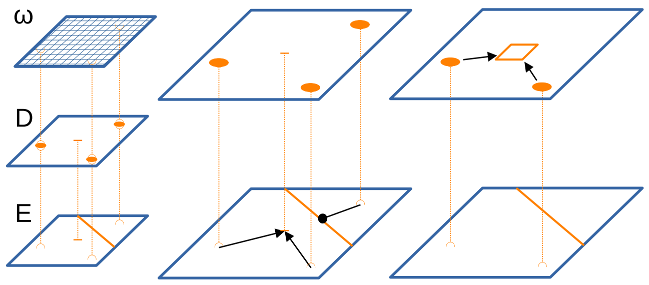

FIGURE 8. Depth upscaling selection kernel. Missing depth in depth map D is interpolated using Gaussian weights ω and boundary edges in E. Unlike the kernel in Figure 7b, the upscaling kernel tests all depth points within the search window.

origin p_o is interpolated according to (3).

$$p_o = \frac{\sum_{\forall p_n \in \mathcal{D}} p_n \omega_n^{\text{gauss}}}{\sum_{m=0}^{|\mathcal{D}|} \omega_m^{\text{gauss}}} \tag{3}$$

Weights ω^{gauss} from an aligned Gaussian kernel are used to prioritize the influence of the nearest known depths within the interpolation window. The upscaled dense depth map $\boldsymbol{D}_2^{\text{src}}$ is composed of the known sparse depths of $\boldsymbol{D}_1^{\text{src}}$ and interpolated depths p_{o} of all positions 'o' that were within kernel range of at least one known depth.

F. FORWARD PROJECTION AND OCCLUDER MASKING

At this stage of the processing pipeline, we have the input views $\boldsymbol{I}^{\text{src}}, \boldsymbol{I}^{\text{trg}}$ and the dense depth map $\boldsymbol{D}_2^{\text{src}}$. The novel view generation and the occluder removal relies on projecting $\boldsymbol{D}_2^{\text{src}}$ to a dense 3D point cloud $\boldsymbol{P}^{3\text{d}}$. The projection is done according to Algorithm 3, based on the pinhole camera model in multiview geometry [18].

Algorithm 3 2D to 3D Projection

```
1: Given depth map $D_2^{src}$, origin $C = [x, y, z]$ (with e.g.
   $C(2) = y$), proj. matrix $\mathcal{P}^{src}$:
2: for each depth $p \in D_2^{src}$ at coord. $(v, u)$ do
3:     $t \leftarrow [u; v; 1]$
4:     $w \leftarrow \text{inverse}(\mathcal{P}) \times t$
5:     $w \leftarrow w./w(4)$
6:     $X \leftarrow \frac{(p-C(3))(w(1)-C(1))}{(w(3)-C(3))} + C(1)$
7:     $Y \leftarrow \frac{(p-C(3))(w(2)-C(2))}{(w(3)-C(3))} + C(2)$
8:     $Z \leftarrow p$
9: end for
```

The dense 3D geometry allows us to specify an "occluder" as any geometry within a specific volume, instead of color-keying or being dependent on green screens. An occluder mask $\boldsymbol{M}^{\text{trg}}$ is created by projecting all 3D points $\boldsymbol{W} = [X; Y, Z]$ within a volume of interest to the image plane of $\boldsymbol{I}^{\text{trg}}$, using (4).

$$\boldsymbol{t}^{\text{trg}} = \mathcal{P}^{\text{trg}} \boldsymbol{W} \text{ , where } \mathcal{P}^{\text{trg}} = [\mathcal{K}^{\text{trg}}][\mathcal{R}|C]^{\text{trg}} \tag{4}$$

The projection matrix of the target view $\mathcal{P}^{\text{trg}}$ is known from a prior offline calibration of corresponding camera's intrinsics $\mathcal{K}$ and extrinsics $\mathcal{R}|\boldsymbol{C}$. A 3-by-3 pixel area around each projected point's coordinates $\boldsymbol{t}^{\text{trg}}$ is included in $\boldsymbol{M}^{\text{trg}}$ to avoid small cracks in the occluder mask. The same projected points are also used to form the depth map $\boldsymbol{D}^{\text{trg}}$. We similarly project $\boldsymbol{P}^{3\text{d}}$ to the image planes of $\boldsymbol{I}^{\text{n1}}, \boldsymbol{I}^{\text{n2}}$ as depth maps $\boldsymbol{D}^{\text{n1}}, \boldsymbol{D}^{\text{n2}}$ to enable subsequent color lookup and view rendering.

The newly projected depth maps tend to have two issues that need to be corrected before view rendering can take place. The first issue is disconnected floating points, projected from $\boldsymbol{P}^{3\text{d}}$ wherever the upscaling process from Section V-E failed to generate a sharp boundary on a depth discontinuity. Such floating points are removed with Algorithm 4, using a search window τ of 5-by-5 pixels and threshold $\tau_{\text{thres}} = 8$. The second issue is the inevitable small cracks that are created due to the change of projection origin from 'src' to 'trg', 'n1', and 'n2'. These cracks are filled via Algorithm 5, which tests whether a given point p_{o} in depth map $\boldsymbol{D}$ is surrounded by significantly lower non-zero depths. If so, p_{o} is likely a small disocclusion, and is replaced by linear interpolation of neighboring depths.

Algorithm 4 Spurious Floating Point Removal

```
1: Given depth map $D$, window $\tau$, threshold $\tau_{thres}$:
2: for each non-zero depth $p_o \in D$ do
3:     for each non-zero $p_n \in \tau$ centered on $p_o$ do
4:         $c \leftarrow c + 1$
5:     end for
6:     if $c < \tau_{thres}$ then
7:         $p_o \leftarrow 0$
8:     end if
9: end for
```

Algorithm 5 Small Crack Detection and Filling

```
1: Given depth map $D$, window $\tau$, threshold $\omega_{thres}$:
2: for each depth $p_o \in D$ do
3:     $p_l \leftarrow$ nearest non-zero depth to left of $p_o$ in $\tau$
4:     $p_r \leftarrow$ nearest non-zero depth to right of $p_o$ in $\tau$
5:     $p_a \leftarrow$ nearest non-zero depth above $p_o$ in $\tau$
6:     $p_b \leftarrow$ nearest non-zero depth below $p_o$ in $\tau$
7:     if $\exists p_l \wedge \exists p_r \wedge |p_l - p_o| > \omega_{thres} \wedge |p_r - p_o| > \omega_{thres}$
       then
8:         $r_l \leftarrow$ coord. distance from $p_o$ to $p_l$
9:         $r_r \leftarrow$ coord. distance from $p_o$ to $p_r$
10:        $p_o \leftarrow r_r p_l/(r_r + r_l) + r_l p_r/(r_r + r_l)$
11:    end if
12:    if
       $\exists p_a \wedge \exists p_b \wedge |p_a - p_o| > \omega_{thres} \wedge |p_b - p_o| > \omega_{thres}$ then
13:        $r_a \leftarrow$ coord. distance from $p_o$ to $p_a$
14:        $r_b \leftarrow$ coord. distance from $p_o$ to $p_b$
15:        $p_o \leftarrow r_a p_b/(r_a + r_b) + r_b p_a/(r_a + r_b)$
16:    end if
17: end for
```

G. COLOR LOOKUP VIA BACKPROJECTION

The last step in generating the $\boldsymbol{I}^{\text{n1}}, \boldsymbol{I}^{\text{n2}}, \boldsymbol{I}_{\text{aug}}^{\text{trg}}$ views is to fetch the per-pixel color information from $\boldsymbol{I}^{\text{src}}$ using the dense depth maps $\boldsymbol{D}^{\text{n1}}, \boldsymbol{D}^{\text{n2}}, \boldsymbol{D}^{\text{trg}}$, in a reverse projection order compared to the depth projection in Section V-F. The color is fetched via reverse projection (a.k.a. reverse lookup) to avoid unnecessary distortion of the image texture. Disocclusion filtering was necessary on the projected depth maps in the forward projection of Section V-F. Performing the same kind of processing on projected color can cause noticeable artefacts, unless more time-consuming texture modeling is employed; reverse-lookup helps to bypass this problem. The lookup process is detailed in Algorithm 6, which combines Algorithm 3 with (4). The disoccluded view $\boldsymbol{I}^{\text{trg}}_{\text{aug}}$ is composed from $\boldsymbol{I}^{\text{trg}}$ and $\boldsymbol{I}^{\text{src}}$ in a manner similar to [28], where only the pixels identified by the occluder mask $\boldsymbol{M}^{\text{trg}}$ are re-painted from $\boldsymbol{I}^{\text{src}}$.

Algorithm 6 Color Lookup via Backprojection

```
 1: Given target depth map $\boldsymbol{D}^{\text{trg}}$, target origin
    $\boldsymbol{C}^{\text{trg}} = [x, y, z]$, target proj. matrix $\mathcal{P}^{\text{trg}}$, source proj.
    matrix $\mathcal{P}^{\text{src}}$, source image $\boldsymbol{I}^{\text{src}}$, empty target image $\boldsymbol{I}^{\text{trg}}$:
 2: for each non-zero depth $p \in \boldsymbol{D}^{\text{trg}}$ at coord. $(v, u)$ do
 3:     $\boldsymbol{t} \leftarrow [u; v; 1]$
 4:     $\boldsymbol{w} \leftarrow \text{inverse}(\mathcal{P}^{\text{trg}}) \times \boldsymbol{t}$
 5:     $\boldsymbol{w} \leftarrow \boldsymbol{w}./\boldsymbol{w}(4)$
 6:     $X \leftarrow \frac{(p-\boldsymbol{C}^{\text{trg}}(3))(\boldsymbol{w}(1)-\boldsymbol{C}^{\text{trg}}(1))}{(\boldsymbol{w}(3)-\boldsymbol{C}^{\text{trg}}(3))} + \boldsymbol{C}^{\text{trg}}(1)$
 7:     $Y \leftarrow \frac{(p-\boldsymbol{C}^{\text{trg}}(3))(\boldsymbol{w}(2)-\boldsymbol{C}^{\text{trg}}(2))}{(\boldsymbol{w}(3)-\boldsymbol{C}^{\text{trg}}(3))} + \boldsymbol{C}^{\text{trg}}(2)$
 8:     $Z \leftarrow p$
 9:     $\boldsymbol{W} \leftarrow [X; Y; Z; 1]$
10:     $\boldsymbol{w}^{\text{src}} \leftarrow \mathcal{P}^{\text{src}}\boldsymbol{W}$
11:     $u^{\text{src}} \leftarrow \boldsymbol{w}^{\text{src}}(1)/\boldsymbol{w}^{\text{src}}(3)$; $v^{\text{src}} \leftarrow \boldsymbol{w}^{\text{src}}(2)/\boldsymbol{w}^{\text{src}}(3)$
12:     $\boldsymbol{I}^{\text{trg}}$ at $(v, u) \leftarrow \boldsymbol{I}^{\text{src}}$ at $(v^{\text{src}}, u^{\text{src}})$
13: end for
```

VI. COMPOSITE VIEW RENDERING

The view rendering stage is the final processing necessary before the original and generated views can be displayed to the user. Each of the five views ($\boldsymbol{I}^{\text{src}}, \boldsymbol{I}^{\text{trg}}, \boldsymbol{I}^{\text{trg}}_{\text{aug}}, \boldsymbol{I}^{\text{n1}}, \boldsymbol{I}^{\text{n2}}$) is in a separate buffer after the main processing has taken place, and their composition into a single render window is done through the Visualization Toolkit (VTK) [40] library's windowing manager. The buffer for each of the input views is, prior to all processing, assigned to a VTK viewport by mapping the buffer contents to a VTK image class bound to the viewport. This ensures that there is no unnecessary overhead during the final render call, beyond sending a content update notification.

The VTK window manager is also responsible for adjusting the on-display resolution of all viewport contents according to particular display resolutions and viewport layout. Due to a quirk of VTK, the buffer contents for each image must be vertically flipped. To avoid the overhead of flipping just prior to rendering, the buffer content is instead reordered during the color lookup stage described in Section V-G by inverting the final indexing of the color assignment in Algorithm 6 step 12 at no extra cost. Since all processing up to this point is completed on full-resolution sensor data, the completion time for all steps in Sections IV and V is independent of the display or viewport settings.

VII. RESULTS

We have recorded a number of sequences in two different indoor environments, visible in Figures 3 and 6, with varying amount of in-scene movement. The hardware specifications are as described in Section III-A, and the image buffer resolution of each view - two original views from the cameras, and three generated views - is set to 1600 $\times$ 1200 pixels, equivalent to the camera sensor resolution. The rendered views are arranged as shown in Figure 1b. For each environment, the camera and lidar parameters were obtained by off-line calibration based on [41], [42], followed by lidar position refinement to reduce projection error. The world coordinate origin is set at the camera recording $\boldsymbol{I}^{\text{src}}$.

A. PROCESSING TIME

The overall rendering frame rate is bounded by the processing described in Section V because it is the limiting factor for updating the rendered content, along with the sensor recording rate (updating rendered content is only necessary if there is any new sensor data). The lidar measurement normalization of Section IV-B does not affect the completion of lidar scan frames at 100 ms intervals, and therefore has no effect on the frame rate. The view generation stages however do have a time cost, which adds up to an average 47 ms per frame, equivalent to approx. 21 frames per second. The timings reported in Figure 9 are in order of execution, and include the CUDA kernel launches, CUDA device synchronization, and any memory copy and set operations as needed for each stage. Most of the processing time is taken by the initial edge extraction and by the depth edge masking steps ($\approx$11 ms each). The initial edge extraction time also includes time needed to transfer the camera views and edge map from host to GPU memory ($\approx$1.5 ms). The final processing stages – dense depth projection and reverse color lookup – take $\approx$10 ms combined for all three generated views.

In particular, the time required to generate a high-resolution depth map from the sparse lidar measurements is approx. 37 ms, with a significantly larger lidar-to-camera resolution upscaling ratio than in e.g. [32]. A larger ratio indicates a larger difference between lidar an camera resolutions, and thus a sparser lidar depth when projected onto the image plane. The performance of several similar methods focused on depth upscaling is shown in Table 1. Due to the differences in scene and capture setups in [30], [32], [36] and our experiment, we show the numbers as reported in [30], [32], [36]. The lidar resolution information in [36] was not mentioned, which we denote by '?' in Table 1. The lidar resolution in Table 1 for our experiment is 240 $\times$ 64, since that is the section of the 360-degree scan that maps into the camera view and is used for the depth upscaling.

The time needed to execute the processing pipeline is not greatly affected by varying the disocclusion size. Table 2 shows the average time per frame from three different scenes, each with a different occluder. The slight increase in computation time is caused by the occluder mask generation and

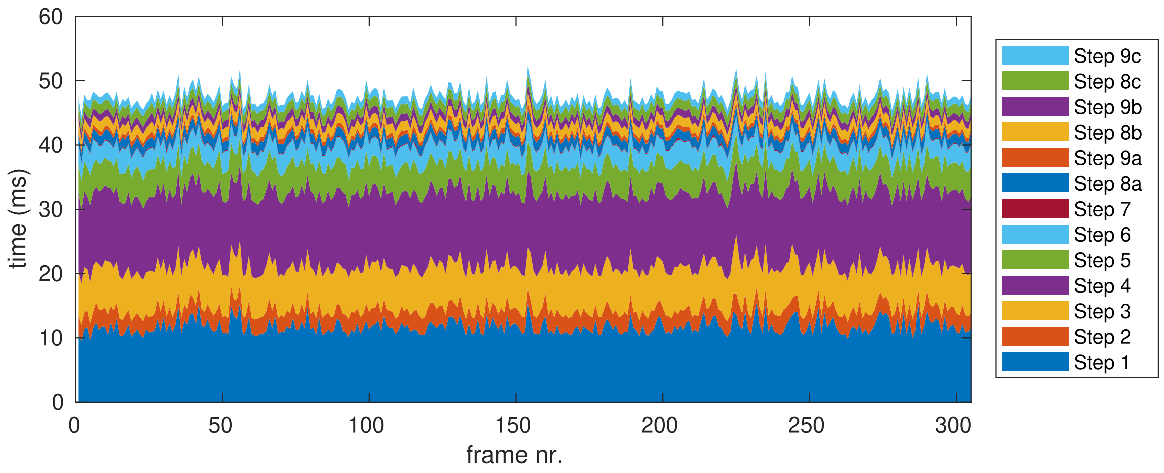

FIGURE 9. Performance breakdown of the proposed pipeline steps. Step 1: Edge extraction, Sec. V-A. Step 2: Sparse projection, Sec. V-B. Step 3: Pierce-through point removal, Sec. V-C. Step 4: Depth edge masking, Sec. V-D. Step 5: Depth upscaling, Sec. V-E. Step 6: Forward projection, Sec. V-F. Step 7: Occluder masking, Sec. V-F. Step 8: Dense depth projection to novel views, Sec. V-F. Step 9: Color lookup for novel views, Sec. V-G. Steps 8 and 9 are repeated for three views, denoted in the legend as "a," "b," and "c."

TABLE 1. Reported performance of depth upscaling from sparse to target resolution, as per [30], [32], [36]. Upscaling ratio indicates the amount of horizontal resolution increase. Time for our method covers stages V-A to V-E.

	Depth res.	Target res.	Upsc. ratio	Time (ms)
[30]	160 × 120	810 × 610	5x	318
[32]	288 × 256	640 × 480	2.3x	8
[36]	?×?	609 × 160	?	150
Ours	240 × 64	1600 × 1200	6.7x	37

TABLE 2. Frame render time (ms) with varying apparent size of disoccluded scene object.

Amount of disoccluded pixels	**2.0%**	**2.9%**	**8.7%**
Avg. total time per frame (ms)	49.6	50.1	51.7

TABLE 3. Amount of missing lidar points in 3 scenes with varying amount of in-scene motion, as percentage of all recorded points.

	No motion	Slight motion	Large motion
Unfiltered	28.0%	5.7%	19.2%
Filtered	24.3%	4.6%	11.9%

TABLE 4. Lidar! (Lidar!) point oscillation amplitude (meters) in scene with no motion. The average, minimum and median amplitude per point is calculated after removing outliers (points with amplitude greater than three scaled median absolute deviations).

	Excluding missing points			Including missing points		
	avg	min	median	avg	min	median
Unfiltered	0.076	0.001	0.066	0.077	0.020	0.080
Filtered	0.039	0.0002	0.034	0.039	0.006	0.043

projection to the composite view, which is increasing the calculation amount as disocclusion increases in scene; the other stages of the processing pipeline are not affected.

B. OUTPUT AND FEATURE ABLATION

1) LIDAR TEMPORAL NORMALIZATION

Part of the perceived quality of a projected image is determined by the stability and consistency of the lidar measurements. This avoids so-called flickering. Stability is measured by the amount of missing point data within the lidar scan of a scene, and consistency is measured by the lidar point oscillation when scanning a static environment. Table 3 lists the amount of missing points from three different scenes - a scene with no motion, a scene with small motion of the robot arm, and a scene with people walking. The unfiltered lidar scans are compared to the scans filtered according to Section IV-B. Our filtering (3) in Section IV-B cannot recover points that are missing throughout the whole recording, e.g. from objects too close to the lidar, but it can recover sporadically excluded points. This is likely why the scene with large motion has the most amount of recovered missing points: when there is more movement in scene, measurements from more rays in more lidar frames are dropped due to the ray crossing an edge of the moving object and failing to get a consistent depth measurement.

The lidar point oscillation, as illustrated in Figure 4a, can lead to noticeable frame-to-frame instability of the projected geometry. Table 4 shows the amplitude of the lidar point oscillation during a recording of a completely static scene, where the amplitude for each point is considered as the distance between the nearest and farthest measurement for each point during the whole recording. For completeness, two comparisons are done, one including "more noisy" points that are sporadically missing during the recording, and one without the missing points, i.e. points that have a measured distance in every lidar frame during the recording. In both cases, the filtering approximately halves the oscillation of distances for most points, reducing the median amplitude from 6.6 and 8 cm to 3.4 and 4.3 cm, respectively.

Another measurement of lidar point stability is shown in Table 5, listing the per-point oscillation between successive frames, which would more directly contribute to frame-by-frame differences in the projected geometry. Here too the filtering approximately halves the point oscillation, by reducing the median frame-to-frame oscillation from 1.4 and 1.6 cm to 0.6 and 0.7 cm, respectively.

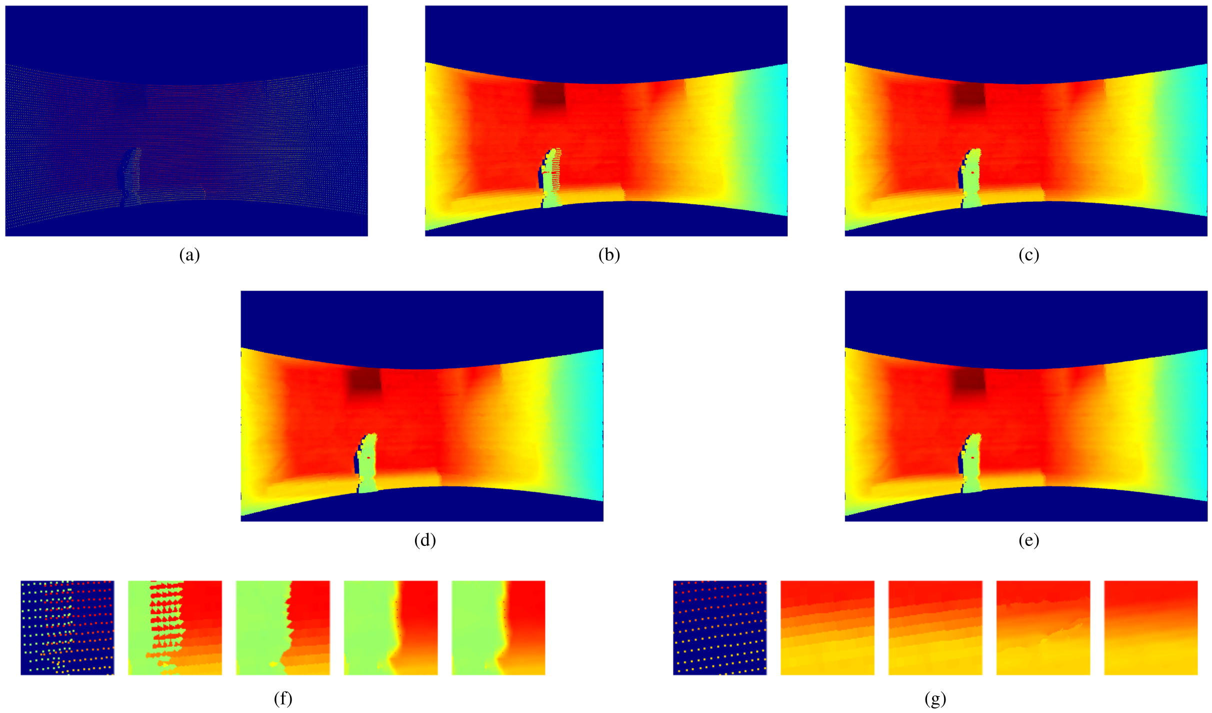

FIGURE 10. The different steps of our depth upscaling. (a) Initial sparse depth (each point upsized from 1 × 1 to 2 × 2 pixels for visibility, best viewed digitally in color); (b) Simple splatting of sparse lidar points; (c) Splatting after pierce-through point removal; (d) Using a color-derived edge map for interpolation bounding; (e) Adding depth-transition masking to edge map; (f) Steps 'a' to 'e' in (cropped) region with background lidar points interleaving with foreground points; (g) Steps 'a' to 'e' in (cropped) region where RGB-derived edges cause separation of smooth-surface depth.

TABLE 5. Lidar! point frame-to-frame oscillation amplitude (meters) in scene with no motion.

	Excluding missing points			Including missing points		
	avg	min	median	avg	min	median
Unfiltered	0.021	0.001	0.014	0.107	0.001	0.016
Filtered	0.015	0.000	0.006	0.081	0.000	0.007

2) DEPTH UPSCALING

As described in Sec. V, our depth upscaling is a combination of several stages. Figure 10 shows how the upscaling results improve with each added stage. The least-effort approach is to simply project the sparse lidar points as-is (see Sec. V-B) with straightforward proximity-based point splatting. Fig. 10b shows the outcome of such an approach. The first notable improvement, shown in Fig. 10c is the addition of pierce-through point removal, as described in Sec. V-C, which prevents the interleaving of foreground and background elements, and approximates an acceptable baseline depth map.

The integration of an edge-bounded nearest-depth interpolation kernel (see Sec. V-A and V-E) is shown in Fig. 10d. If the edges in color image correspond to object boundaries, then this may be sufficient. However as demonstrated in [31], over-segmentation of the edge map is a significant risk and tends to result in excessive interpolation blocking. Such over-segmentation leads to "missing" depth points in between valid lidar measurements, where all nearby measurements in the interpolation kernel are blocked by an edge. The depth-transition masking stage (see Sec. V-D) is added to remedy such an edge over-segmentation, resulting in the depth map shown in Fig. 10e. The top and bottom regions of the depth map are left unfilled, as no lidar measurements are within acceptable proximity to provide support for depth estimation.

3) OCCLUDER REPLACEMENT

The upscaled depth is used as projection basis for re-painting the occluder-occupied pixels with data from another recorded view. Figure 11b shows the repainting outcome of our approach. For comparison, we show two other approaches that might be considered as alternatives, direct view replacement and patch shifting based on single measurement (described below). For the sake of comparison, we assume that color-keying can be used to identify the occluder mask in these alternatives.

The direct view replacement implies a direct copy of pixel data from the other view without any projective transform, and is shown in Figure 11c. It has next to no performance cost, since it is merely a buffer copy operation. However, it is not a feasible solution in terms of producing a view with a consistent geometry because it is highly unlikely that the same 3D background geometry projects to the exact same pixel coordinates in two cameras. Patch shifting, shown in Figure 11d, uses a single central lidar measurement to determine the horizontal and vertical disparity for shifting the entire disocclusion patch in from the other view. The selected patch therefore has uniform disparity throughout. Under a rectified camera setup and sufficiently flat background, this

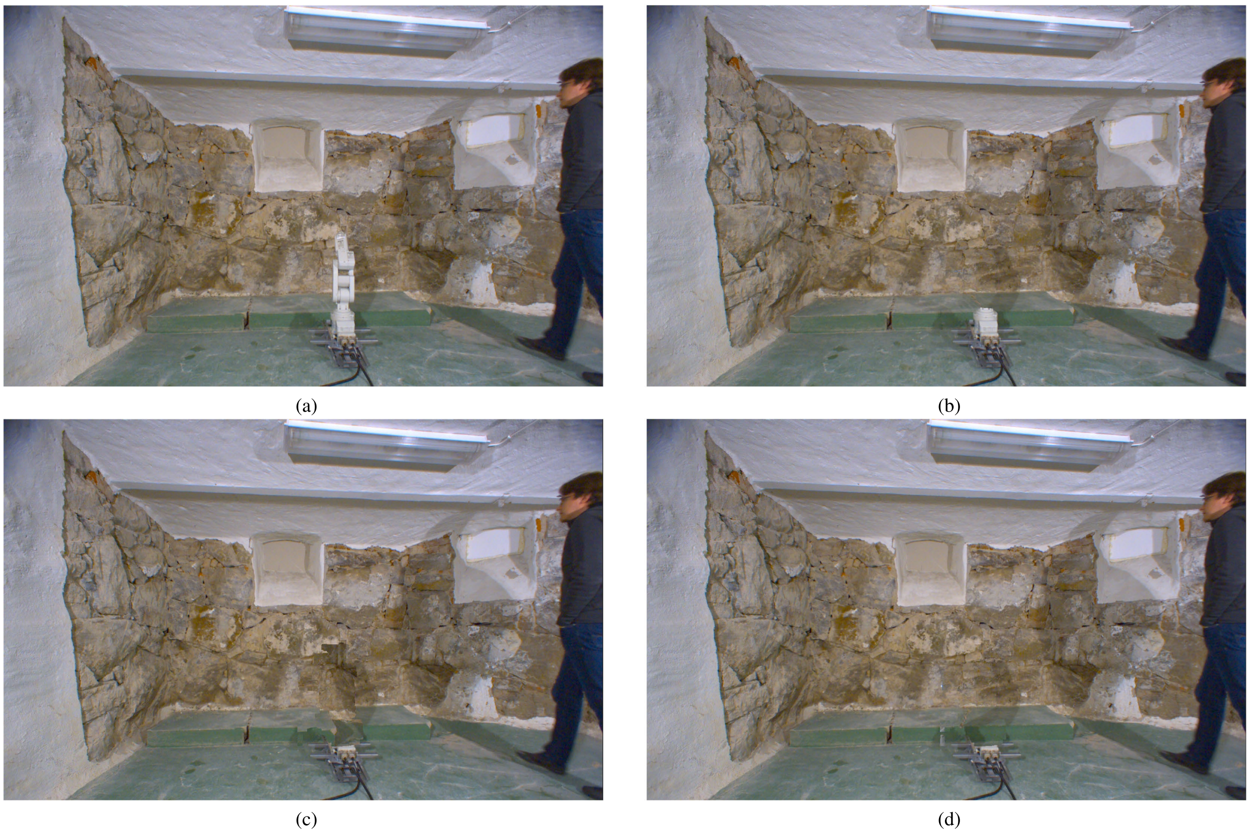
(a) (b) (c) (d)

FIGURE 11. **Results of our occluder replacement (b) compared to original view (a), color-keying based view replacement (c) and patch shifting based on single depth disparity (d).**

may be a useful approach. However, with non-rectified views the consistency of the revealed geometry is less accurate, especially for surfaces that are on a depth gradient and thus inconsistent with a uniform patch-wide disparity; this can be seen on the skewed shadows behind the object in Figure 11d. However, such uniform patch shifting does allow to repaint areas lying outside the range of lidar beam spread - hence why the base of the object is not wholly repainted in Figure 11b, but is repainted in Fig. 11c and 11d.

For mining applications, using off-the-shelf solutions from computer vision libraries can be preferable to adapting purpose-built solutions. Therefore, we also show a comparison lidar-based view generation by a simple direct lidar meshing and texture projection in Figure 12 afforded by the VTK function *'projectedTexture'*. In this approach, the sparse lidar points are used as mesh vertices without smoothing, and the resulting mesh is used as the projection surface. Missing lidar points are excluded, and irregular gaps between vertices are filled in best-effort basis, connecting to nearest neighbors within a limited proximity. It is worth noting that as-is, VTK's texture projection does not handle occlusion or projection shadowing (the consequence of which is shown in Figure 12c, and would require considerable rework.

VIII. DISCUSSION

The context of the mining industry made us focus on designing a pipeline that avoids any sort of pre-recorded content or view recreation from trained dictionaries. It cannot be denied that the bulk of research activity related to RGB-D data nowadays links to machine learning for depth (and view) creation. However, following that path would have countered the context in which we place our work, where presenting real-looking yet inaccurate view is inherently dangerous. The decisions in designing the presented pipeline are therefore focused on using and relying on the data actually recorded by the camera and lidar sensors.

The lidar temporal filtering was considered a necessary component even though it relies on smoothing over time. The oscillation and random dropout of supposedly static points from the tested lidar led to inconsistencies from frame to frame, which implies flickering in the presented views. In an applied context, this could lead to operator distraction or misinterpretation of the displayed scene.

The requirement of real-time performance directed the depth upscaling component to the straightforward interpolation method described in [32]. An optimization-based depth diffusion similar to [30], [33], [37] would likely have reached higher quality, but at the cost of more milliseconds per frame. At the time of writing, the CUDA sparse LS-solver implementation required to implement [31] was only available via host execution, i.e. processing on CPU rather than the GPU, which compromises the real-time feasibility of that approach. Therefore, the upscaling method, inspired by [32], was selected and enhanced by both widening the edge-finding

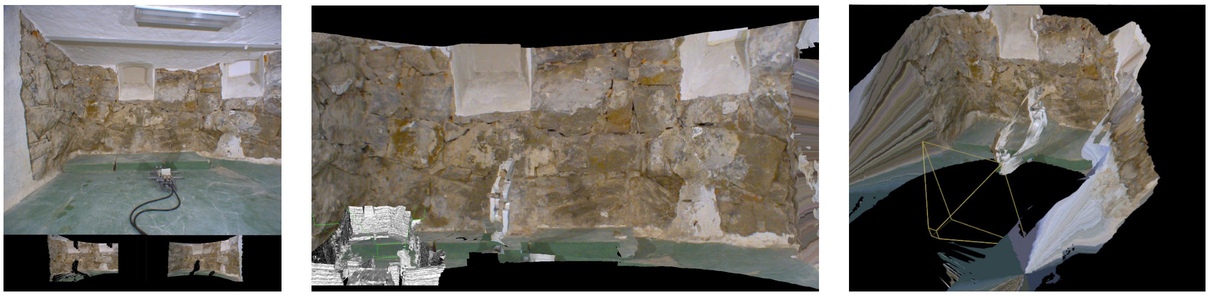
(a) (b) (c)

FIGURE 12. (a) Results of our view projection (a), 47ms for three views. (b) Results of lidar meshing and VTK *projectedTexture* method, approx. 82 ms for single view. (c) Shows the problem of lacking self-occlusion in the *projectedTexture* projection in (b), causing the target object to be visible on foreground and background geometry.

marching ray, as well as pre-filtering the reference edge map with discontinuities in sparse depth.

Each step of the processing pipeline is developed as a separate set of CUDA kernels, operating on numeric data in general-purpose buffers. There may be further performance gains by combining the stages, and by moving the non-accelerated parts such as OpenCV's edge detection onto the GPU. At this point, the rendered viewport layout uses VTK for adaptability; in case of a specific application with a specific viewport design, the VTK framework can be removed and the separate view buffers can be combined during view generation. However, that would require moving outside the scope of this feasibility study, and involve UX design testing with specific operators for specific tasks.

IX. CONCLUSION AND FUTURE WORK

In this study, we have focused on demonstrating the real-time feasibility of augmented rendering towards the mining context, by presenting an end-to-end system that covers the whole processing chain, from sensors to the rendered views. A key goal behind the system design was to avoid shortcuts such as green screens, pre-recorded reference databases, or content inpainting based on learned priors. The use of such priors may have increased the apparent render quality. However, the goal of this work was to investigate the feasibility and technical readiness of augmented remote operation for the mining industry, where presenting incorrect or misleading content to an operator may do more harm than good. As such, the solutions presented in Sections III through VI are designed for handling the live sensor data within the real-time limits imposed by said sensors. While our system does not achieve as low render times as shown by [20], we avoid some of the limitations imposed therein, such as the greenscreen keying and background removal, and demonstrate an alternate approach that still attains real-time performance within the constraints outlined in [5].

In other non-entertainment contexts, augmented remote operation appears to benefit the operators in terms of their Quality of Experience (QoE) and task accomplishment, as demonstrated in [6], [8]–[11]. The augmentation covered by our system is a relatively simple view manipulation, meant to explore the technical feasibility more than the specific operator experience. Future work encompasses the assessment of our system from a QoE perspective, with a more direct focus on the interface layout and the visual design of augmentations tailored to specific mining tasks.

ACKNOWLEDGMENT

The authors would like to thank J. Edlund at Mid Sweden University for assistance in setting up the test environment and contributing to the remote-control codebase for the experimental demonstrator, L. Önnerlöv at Boliden Minerals AB, and A. Nilsson at Jama Mining Machines AB for providing the mining industry's perspectives and insights on experimental setup and practical considerations.

COMPLIANCE WITH ETHICAL STANDARDS

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

REFERENCES

[1] G. Carra, A. Argiolas, A. Bellissima, M. Niccolini, and M. Ragaglia, "Robotics in the construction industry: State of the art and future opportunities," in *Proc. 35th Int. Symp. Autom. Robot. Construction (ISARC)*, Jul. 2018, pp. 1–8.

[2] F. Sánchez and P. Hartlieb, "Innovation in the mining industry: Technological trends and a case study of the challenges of disruptive innovation," *Mining, Metall. Explor.*, vol. 37, pp. 1385–1399, Jul. 2020.

[3] C. Sganzerla, C. Seixas, and A. Conti, "Disruptive innovation in digital mining," *Procedia Eng.*, vol. 138, pp. 64–71, Jan. 2016.

[4] P. Tripicchio, E. Ruffaldi, P. Gasparello, S. Eguchi, J. Kusuno, K. Kitano, M. Yamada, A. Argiolas, M. Niccolini, M. Ragaglia, and C. A. Avizzano, "A stereo-panoramic telepresence system for construction machines," *Procedia Manuf.*, vol. 11, pp. 1552–1559, Jan. 2017.

[5] Y. Yun, S. J. Lee, and S.-J. Kang, "Motion recognition-based robot arm control system using head mounted display," *IEEE Access*, vol. 8, pp. 15017–15026, 2020.

[6] K. Brunnström, E. Dima, T. Qureshi, M. Johanson, M. Andersson, and M. Sjöström, "Latency impact on quality of experience in a virtual reality simulator for remote control of machines," *Signal Process., Image Commun.*, vol. 89, Nov. 2020, Art. no. 116005.

[7] V. Kohn and D. Hardborth, "Augmented reality—A game changing technology for manufacturing processes," in *Proc. 26th Eur. Conf. Inf. Syst. (ECIS)*, 2018, pp. 1–18.

[8] E. Dima, K. Brunnström, M. Sjöström, M. Andersson, J. Edlund, M. Johanson, and T. Qureshi, "Joint effects of depth-aiding augmentations and viewing positions on the quality of experience in augmented telepresence," *Qual. User Exper.*, vol. 5, no. 1, pp. 1–7, Dec. 2020.

[9] F. Bruno, A. Lagudi, L. Barbieri, D. Rizzo, M. Muzzupappa, and L. De Napoli, "Augmented reality visualization of scene depth for aiding ROV pilots in underwater manipulation," *Ocean Eng.*, vol. 168, pp. 140–154, Nov. 2018.

[10] B. Bejczy, R. Bozyil, E. Vaičekauskas, S. B. K. Petersen, S. Bøgh, S. S. Hjorth, and E. B. Hansen, "Mixed reality interface for improving mobile manipulator teleoperation in contamination critical applications," *Procedia Manuf.*, vol. 51, pp. 620–626, Jan. 2020.

[11] B. P. Vagvolgyi, W. Pryor, R. Reedy, W. Niu, A. Deguet, L. L. Whitcomb, S. Leonard, and P. Kazanzides, "Scene modeling and augmented virtuality interface for telerobotic satellite servicing," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 4241–4248, Oct. 2018.

[12] G. Mastrorocco, R. Salvini, and C. Vanneschi, "Fracture mapping in challenging environment: A 3D virtual reality approach combining terrestrial LiDAR and high definition images," *Bull. Eng. Geol. Environ.*, vol. 77, no. 2, pp. 691–707, May 2018.

[13] L. Chun-Lei, S. Hao, L. Chun-Lai, and L. Jin-Yang, "Intelligent detection for tunnel shotcrete spray using deep learning and LiDAR," *IEEE Access*, vol. 8, pp. 1755–1766, 2020.

[14] F. Okura, M. Kanbara, and N. Yokoya, "Augmented telepresence using autopilot airship and omni-directional camera," in *Proc. IEEE Int. Symp. Mixed Augmented Reality*, Oct. 2010, pp. 259–260.

[15] C. Xue Er Shamaine, Y. Qiao, J. Henry, K. McNevin, and N. Murray, "RoSTAR: ROS-based telerobotic control via augmented reality," in *Proc. IEEE 22nd Int. Workshop Multimedia Signal Process. (MMSP)*, Sep. 2020, pp. 1–6.

[16] B. Omarali, B. Denoun, K. Althoefer, L. Jamone, M. Valle, and I. Farkhatdinov, "Virtual reality based telerobotics framework with depth cameras," in *Proc. 29th IEEE Int. Conf. Robot Human Interact. Commun. (RO-MAN)*, Aug. 2020, pp. 1217–1222.

[17] D. Lee and Y. S. Park, "Implementation of augmented teleoperation system based on robot operating system (ROS)," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 5497–5502.

[18] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[19] W. Sun, L. Xu, O. C. Au, S. H. Chui, and C. W. Kwok, "An overview of free view-point depth-image-based rendering (DIBR)," in *Proc. APSIPA Annu. Summit Conf.*, 2010, pp. 1023–1030.

[20] S. Rasmuson, E. Sintorn, and U. Assarsson, "A low-cost, practical acquisition and rendering pipeline for real-time free-viewpoint video communication," *Vis. Comput.*, vol. 37, no. 3, pp. 553–565, 2020.

[21] S. Meerits, V. Nozick, and H. Saito, "Real-time scene reconstruction and triangle mesh generation using multiple RGB-D cameras," *J. Real-Time Image Process.*, vol. 16, no. 6, pp. 2247–2259, Dec. 2019.

[22] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," *Proc. SPIE*, vol. 5291, pp. 93–104, May 2004.

[23] C. Vázquez, W. J. Tam, and F. Speranza, "Stereoscopic imaging: Filling disoccluded areas in depth image-based rendering," *Proc. SPIE*, vol. 6392, Oct. 2006, Art. no. 63920D.

[24] O. Elharrouss, N. Almaadeed, S. Al-Maadeed, and Y. Akbari, "Image inpainting: A review," *Neural Process. Lett.*, vol. 51, no. 2, pp. 2007–2028, 2019.

[25] H. Dhamo, K. Tateno, I. Laina, N. Navab, and F. Tombari, "Peeking behind objects: Layered depth prediction from a single image," *Pattern Recognit. Lett.*, vol. 125, pp. 333–340, Jul. 2019.

[26] S. M. Muddala, R. Olsson, and M. Sjöström, "Disocclusion handling using depth-based inpainting," in *Proc. 5th Int. Conf. Adv. Multimedia, (MMEDIA)*, Venice, Italy: International Academy, Research and Industry Association (IARIA), Apr. 2013, pp. 136–141.

[27] S. Li, C. Zhu, and M.-T. Sun, "Hole filling with multiple reference views in DIBR view synthesis," *IEEE Trans. Multimedia*, vol. 20, no. 8, pp. 1948–1959, Aug. 2018.

[28] M.-L. Wu and V. Popescu, "RGBD temporal resampling for real-time occlusion removal," in *Proc. ACM SIGGRAPH Symp. Interact. 3D Graph. Games*, May 2019, pp. 1–9.

[29] S. Yang, B. Li, M. Liu, Y.-K. Lai, L. Kobbelt, and S.-M. Hu, "HeteroFusion: Dense scene reconstruction integrating multi-sensors," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 11, pp. 3217–3230, Nov. 2020.

[30] D. Ferstl, C. Reinbacher, R. Ranftl, M. Ruether, and H. Bischof, "Image guided depth upsampling using anisotropic total generalized variation," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 993–1000.

[31] S. Schwarz, M. Sjostrom, and R. Olsson, "A weighted optimization approach to time-of-flight sensor fusion," *IEEE Trans. Image Process.*, vol. 23, no. 1, pp. 214–225, Jan. 2014.

[32] H. Plank, G. Holweg, T. Herndl, and N. Druml, "High performance time-of-flight and color sensor fusion with image-guided depth super resolution," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 1213–1218.

[33] F. S. Zakeri, M. Sjöström, and J. Keinert, "Guided optimization framework for the fusion of time-of-flight with stereo depth," *J. Electron. Imag.*, vol. 29, no. 5, Oct. 2020, Art. no. 053016.

[34] M. Ni, J. Lei, R. Cong, K. Zheng, B. Peng, and X. Fan, "Color-guided depth map super resolution using convolutional neural network," *IEEE Access*, vol. 5, pp. 26666–26672, 2017.

[35] R. Chen and W. Gao, "Color-guided depth map super-resolution using a dual-branch multi-scale residual network with channel interaction," *Sensors*, vol. 20, no. 6, p. 1560, Mar. 2020.

[36] C. S. Weerasekera, T. Dharmasiri, R. Garg, T. Drummond, and I. Reid, "Just-in-time reconstruction: Inpainting sparse maps using single view depth predictors as priors," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 1–9.

[37] P. Harish, V. Vineet, and P. Narayanan, "Large graph algorithms for massively multithreaded architectures," Int. Inst. Inf. Technol. Hyderabad, Hyderabad, India, Tech. Rep. IIIT/TR/2009/74, 2009.

[38] Y. Song, H. Zhang, Y. Liu, J. Liu, H. Zhang, and X. Song, "Background filtering and object detection with a stationary LiDAR using a layer-based method," *IEEE Access*, vol. 8, pp. 184426–184436, 2020.

[39] G. Bradski and A. Kaehler, "Learning OpenCV: Computer vision with the OpenCV library," O'Reilly Media, 2008.

[40] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit*, 4th ed. New York, NY, USA: Kitware, 2006.

[41] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000.

[42] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 1997, pp. 1106–1112.

ELIJS DIMA (Graduate Student Member, IEEE) was born in Liepaja, Latvia, in 1990. He received the B.Sc. and M.Sc. degrees in computer engineering and the Licentiate degree in computer and system sciences from Mid Sweden University, Sundsvall, Sweden, in 2013, 2015, and 2018, respectively, where he is currently pursuing the Ph.D. degree in computer and system sciences with the Realistic 3D Research Group.

Since 2015, he has been an Undergraduate Supervisor and a Course Assistant with the Department of Information and Communication Systems, Mid Sweden University. His research interests include multi-view and light field capture and rendering, synchronization, calibration, modeling, data fusion and capture operation of multi-camera systems and mixed-sensor systems, integration and application of such systems, real-time visual presentation through augmented reality, telepresence, the combination thereof, the quality of experience, and technical design aspects for such systems with relation to industrial contexts.

MÅRTEN SJÖSTRÖM (Senior Member, IEEE) received the M.Sc. degree in electrical engineering and applied physics from Linköping University, Sweden, in 1992, the Licentiate of Technology degree in signal processing from the KTH Royal Institute of Technology, Stockholm, Sweden, in 1998, and the Ph.D. degree in modeling of nonlinear systems from EPFL, Lausanne, Switzerland, in 2001.

He was an Electrical Engineer with ABB, Sweden, from 1993 to 1994, and a Fellow with the CERN, from 1994 to 1996. In 2001, he joined Mid Sweden University, where he was appointed as an Associate Professor and a Full Professor in signal processing, in 2008 and 2013, respectively. He founded the Realistic 3D Research Group, in 2007. He has been the Head of the Research Subject Computer Engineering with Mid Sweden University, since 2013. His current research interests include multidimensional signal processing and imaging, system modeling, and identification.