Efficient 3D Reconstruction, Streaming and Visualization of Static and Dynamic Scene Parts for Multi-client Live-telepresence in Large-scale Environments

Leif Van Holland\textsuperscript{1} Patrick Stotko\textsuperscript{1} Stefan Krumpen\textsuperscript{1} Reinhard Klein\textsuperscript{1} Michael Weinmann\textsuperscript{2}

\textsuperscript{1}University of Bonn \hspace{1cm} \textsuperscript{2}Delft University of Technology

Figure 1. Visualization of the key components of our proposed pipeline. Color and depth images are blended with class and instance information, and shown along with the optical flow w.r.t. to the previous frame (first image). This information is integrated to produce a mask that segments the frame into static and dynamic regions (second image). Together with an accumulated 3D motion estimate (third image), the scene is streamed to one or multiple remote clients for immersive exploration in VR (fourth image).

Abstract

Despite the impressive progress of telepresence systems for room-scale scenes with static and dynamic scene entities, expanding their capabilities to scenarios with larger dynamic environments beyond a fixed size of a few square meters remains challenging.

In this paper, we aim at sharing 3D live-telepresence experiences in large-scale environments beyond room scale with both static and dynamic scene entities at practical bandwidth requirements only based on light-weight scene capture with a single moving consumer-grade RGB-D camera. To this end, we present a system which is built upon a novel hybrid volumetric scene representation in terms of the combination of a voxel-based scene representation for the static contents, that not only stores the reconstructed surface geometry but also contains information about the object semantics as well as their accumulated dynamic movement over time, and a point-cloud-based representation for dynamic scene parts, where the respective separation from static parts is achieved based on semantic and instance information extracted for the input frames. With an independent yet simultaneous streaming of both static and dynamic content, where we seamlessly integrate potentially moving but currently static scene entities in the static model until they are becoming dynamic again, as well as the fusion of static and dynamic data at the remote client, our system is able to achieve VR-based live-telepresence at interactive rates. Our evaluation demonstrates the potential of our novel approach in terms of visual quality, performance, and ablation studies regarding involved design choices.

1. Introduction

Sharing immersive, full 3D experiences with remote users, while allowing them to explore the respectively shared places or environments individually and independently from the sensor configuration, represents a core element of metaverse technology. Beyond pure 2D images or 2D videos, 3D telepresence is defined as the impression of individually being there in an environment that may differ from the user’s actual physical environment [20, 28, 41, 84, 152]. This offers new opportunities for diverse applications including remote collaboration, entertainment, advertisement, teaching, hazard site exploration, rehabilitation as well as for joining virtual sports events, work meetings, remote maintenance/consulting or simply enjoying social gatherings. In turn, the possibilities for virtually bringing people or experts together from all over the world in a digital twin of a location as well as the live-virtualization of such environments and events may reduce the effort regarding on-site traveling for many people, which not only helps to reduce our CO\textsubscript{2} footprint and save time but also facilitates economically less well-situated or handicapped people to access such events.

The creation of an immersive telepresence experience relies on various factors. Respective core features are visu-
ally convincing depictions of a scenario as well as the subjective experience, vividness and interactivity in terms of operating in the scene [117, 122]. Therefore, the involved aspects include display parameters (e.g., resolution, frame rate, contrast, etc.), the presentation of the underlying data, its consistency, low-latency control to avoid motion sickness, the degree of awareness and the suitability of controller devices [20, 28, 41, 84, 117, 122, 152]. Furthermore, experiencing 3D depth cues like stereopsis, motion parallax, and natural scale also contribute to the perceived level of immersion and copresence [35, 88].

However, such immersive 3D scene exploration experiences become particularly challenging for telepresence in live-captured environments due to the additional requirement of accurately reconstructing the digital twin of the underlying scene on the fly as well as its efficient streaming and visualization to remote users under the constraints imposed by available network bandwidth and client-side compute hardware. Among many approaches, impressive immersive AR/VR-based live-3D-telepresence experiences have only been achieved based on advanced RGB-D acquisition for dynamic scenes on room scale using special expensive static capture setups [14, 16, 21, 26, 37, 52, 64, 68, 78, 79, 97, 98, 112, 129, 144, 170] and display technology [68], as well as for static scenes beyond room scale operating in the scene [117, 122]. For the latter category, bandwidth requirements have been reduced from hundreds of MBit/s for a single user [87] to around 15MBit/s for group-scale sharing of telepresence live-captured environments while also handling network interruptions [123, 124, 126], thereby even allowing live-teleoperation of robots [125]. However, expanding the capabilities and, thereby, overcoming the aforementioned limitations in large dynamic environments for many users with low-cost setups still remains an open challenge.

In this paper, we aim at sharing 3D live-telepresence experiences in large-scale environments beyond room scale with both static and dynamic scene entities at practical bandwidth requirements and based on light-weight scene capture with a single moving consumer-grade RGB-D camera. For this purpose, we propose a respective system that relies on efficient 3D reconstruction, streaming and immersive visualization for dynamic large-scale scenes as depicted in Figure 1.

In particular, the key contributions of our work are:

- For the sake of efficiency, our system leverages a hybrid volumetric scene representation, where we use optical flow and instance information extracted from the input frames to detect static and dynamic scene entities, thereby allowing the combination of a classic implicit surface geometry representation enriched with the object semantics as well as their accumulated dynamic motion over time, with a point-cloud-based representation of dynamic parts.
- We achieve efficient data streaming to remote users by the separate yet simultaneous streaming of both static and dynamics scene information, where we seamlessly integrate potentially moving but currently static scene entities in the static model until they are becoming dynamic again. Additionally, the fusion of static and dynamic data at the remote client allows VR-based visualization of the scene at interactive rates.
- We demonstrate the potential of our approach in the scope of several experiments and provide an ablation study for respective design choices.

Furthermore, while not being among the main contributions of our work, our approach also inherits the robustness of previous techniques to network interruptions for the reconstruction of the static scene parts as well as the scalability to group-scale telepresence [123, 125, 126].

2. Related Work

Telepresence Systems Despite almost two decades of developments, the development of systems that allow immersive telepresence experiences remains challenging due to the prerequisite of simultaneously achieving high-fidelity real-time 3D scene reconstruction, the efficient streaming and management of the reconstructed models and the high-quality visualization based on AR and VR equipment. Early approaches were limited by the capabilities of the available hardware [31, 58, 66, 90, 134, 140], inaccurate silhouette-based reconstruction techniques [76, 105]. Depth-based 3D scanning led to improved reconstruction quality and allowed telepresence at room [32, 47, 54, 78, 80, 85], however, remaining artifacts induced by the high sensor noise and temporal inconsistency in the reconstruction process still impacted the visual experience. More recently, advances in 3D scene capture, streaming and visualization technology led to impressive immersive AR/VR-based live-3D-telepresence experiences. Live-telepresence for small-scale scenarios of a few squaremeters has been achieved based on light-weight capture setups for teleconferencing [4, 13, 24, 53, 94, 101] and other collaborative scenarios [25, 36, 77, 121, 136, 163] as well as based on expensive multi-camera static and pre-calibrated capture setups [14, 16, 21, 26, 37, 52, 64, 68, 78, 79, 97, 98, 112, 129, 144, 170]. Furthermore, live-telepresence for scenarios beyond room scale has been achieved based on low-cost and light-weight incremental scene capture with a moving depth camera [6, 87, 123–126, 160], allowing remote users to immersively explore a live-captured environment independent from the sensor configurations. Regarding the latter approaches, impractical bandwidth requirements of up to 175MBit/s for
immersively exploring large-scale environments by a single user [87] have been overcome by more recent approaches that allow group-scale sharing of telepresence experiences in live-captured environments and handling network interruptions [123–126] as well as live-teleoperation of robots [125]. Furthermore, mechanisms for annotation, distance measurement [125] and efficient collaborative VR-based 3D labeling were added [169]. However, practical sharing of live-captured 3D experiences in dynamic large-scale environments for many users with low-cost setup still remains an open challenge. The same applies for immersive robot teleoperation where approaches focused on small-scale scenarios with dynamics [65, 74, 91, 104, 114, 125, 139, 150] and large-scale, static scenarios [125].

In contrast to the aforementioned approaches, we propose a live-telepresence system for large-scale environments beyond room-scale and including scene dynamics.

3D Reconstruction and SLAM Techniques Current state-of-the-art telepresence systems rely on depth-based simultaneous localization and mapping (SLAM) techniques. Examples are the use of depth-sensor-based 3D scene capture based on surfels [43] or extensions of KinectFusion [48, 93] in terms of voxel block hashing techniques [55–57, 95, 106] for incremental scene capture for large-scale telepresence applications [87, 123–126]. To avoid the need for depth sensors, more recent Simultaneous Localization and Mapping (SLAM) approaches for incremental scene capture - that might be applicable in respective telepresence applications - leveraged principles of deep learning [17, 60, 63, 67, 151, 156, 157]. Further approaches investigated 3D reconstruction from multiple synchronized cameras [1, 2, 23, 46, 86].

Recently, neural scene representation and rendering techniques [137, 138] have led to significant improvements in reconstruction quality for small-scale objects or scenes. The underlying idea originates from novel view synthesis and consists of training a neural network to represent a scene with its weights, so that respectively synthesized views match the input photographs. In particular, this includes implicit scene representations based on NeRFs [83] and respective extensions towards speeding up model training [7, 11, 18, 29, 89, 110, 132] with training times of seconds, the adaptation to constrained image collections [10, 81], deformable scenes [8, 33, 75, 96, 99, 100, 102, 107, 109, 141, 142] and video inputs [22, 34, 71, 72, 103, 154], the refinement or complete estimation of camera pose parameters for the input images [50, 73, 82, 130, 131, 147, 159, 168], combining NeRFs with semantics regarding objects in the scene [30, 145, 166], incorporating depth cues [3, 18, 111, 113, 148] to guide the training and allow handling textureless regions, handling large-scale scenarios [133, 143], and streamable representations [12]. However, despite promising results, further improvements regarding efficiency are required for the joint camera pose estimation and neural scene reconstruction [131, 168] as required in a SLAM setting to achieve beyond the reported 5 fps on a current high-quality GPU (Nvidia RTX 3090) [168] while also reducing the jittering of the depicted scene during exploration.

Particularly addressing dynamic environments, various approaches focused on filtering dynamic objects and only reconstructing the static background [5, 27, 61, 118, 155, 161, 164] or additionally reconstructing the dynamics based on rigid object tracking and reconstruction [42, 70, 115, 116, 128, 153] and non-rigid object tracking and reconstruction [19, 38, 45, 49, 59, 69, 92, 119, 120, 127, 149, 149, 158, 162]. Taking inspiration of the non-rigid scene reconstruction approaches in terms of separating static and dynamic scene parts, the 3D reconstruction approach involved in our live-telepresence system is particularly designed for capturing large-scale environments (i.e., beyond scenarios limited to a small area of a few squaremeters) with both static and dynamic entities based on a single moved RGB-D camera. Our hybrid volumetric scene representation leverages semantic and instance information to detect dynamic scene entities and combines a voxel-based scene representation for the static parts, where we also accumulate information on whether and how significant objects have been moved, with a point-cloud-based representation of dynamic parts. A major contribution of our work is the separate but simultaneous streaming of both static and dynamics scene information and its VR-based visualization at interactive rates.

3. Methodology

As shown in Figure 2, our live-telepresence system for large-scale environments with scene dynamics at practical bandwidth requirements takes a continuous stream of RGB-D images \((I_1, D_1), (I_2, D_2), \ldots\) from a moving depth camera as input, where \(I_k(u) \in \mathbb{R}^3\) represents the red, green and blue color values of frame \(k\), and \(D_k(u) \in \mathbb{R}\) the corresponding raw depth measurement at pixel \(u \in U \subseteq \mathbb{N}^2\), with \(U\) being the image domain. The main challenge consists in efficiency when processing these measurements, integrating them into a consistent model and streaming the latter over the network at practical bandwidth requirements to remote clients, where it has to be visualized at adequate visual quality at tolerable overall latency. For this purpose, we use a hybrid scene representation that separately handles static and dynamic scene parts, thereby allowing the combination of efficient large-scale 3D scene mapping techniques, that face problems with dynamic regions, with efficient point-based reconstruction for the dynamic parts. In more detail, we segment the frames of the input stream into static and dynamic regions by determining score maps \(S_k\), where \(S_k(u) \in \mathbb{R}\) describes the
amount of dynamicity in frame $k$ at pixel $u$. This separation allows us to efficiently reconstruct, stream and immersively visualize static regions using existing state-of-the-art large-scale telepresence techniques [123, 126] while simultaneously reconstructing, streaming and visualizing dynamic scene parts based on a point-based representation in terms of a partial RGB-D image and its corresponding estimated camera pose, thereby limiting the amount of data to be transferred and reducing the processing time. After streaming the hybrid scene representation to remote users, its static and dynamic parts are joined in a combined 3D visualization. In the following subsections, we provide more details on the different steps of our pipeline.

3.1. Segmentation into Static and Dynamic Regions

For the sake of efficiency, we segment the RGB-D frames of the input stream into static and dynamic regions, which will later allow the efficient treatment of the different types of scene parts. For this purpose, we compute score maps $S_k$, where $S_k(u) \in \mathbb{R}$ describes the amount of dynamicity in frame $k$ at pixel $u$. In the following, we will assume that these scores are normalized in the sense that a pixel is deemed static if $S_k(u) \leq 1$, and dynamic if $S_k(u) > \tau$, where $\tau \geq 1$ is a threshold that allows for a region of uncertainty between the static and dynamic labels. To compute the dynamicity score $S_k$ of frame $k$, we first detect objects in $I_k$ using instance segmentation, which yields both a class label and an instance ID for each pixel in the image, i.e. $(L_k, t_k) = f_{seg}(I_k)$ of $I_k$, where $L_k(u) \in \mathbb{N}$ is the predicted class label and $t_k(u) \in \mathbb{N}$ is the instance ID at pixel $u$. The raw output of the segmentation network may consist of multiple, potentially overlapping region proposals which we integrate into the instance and labels maps using non-maximum suppression. Pixels without any proposal, belong to a low confidence detection or to a region with pixel count below a certain threshold are ignored by setting $L_k(u) = t_k(u) = 0$. See the supplemental material for a detailed explanation of this procedure. In our experiments, we used SoloV2 [146] using pretrained weights from [9].

Next, we estimate the backward optical flow $F_k = f_{flow}(I_k, I_{k-1})$, where $F_k(u) \in \mathbb{R}^2$ is the corresponding flow vector at pixel $u$, such that $u$ in $I_k$ corresponds to $u + F_k(u)$ in $I_{k-1}$. For $f_{flow}$ we used a pretrained version [15] of LiteFlowNet2 [44].

Subsequently, we estimate the camera motion

$$\xi_k = f_{pose}(I_{k-1}, I_k, D_{k-1}, D_k) \in se(3) \quad (1)$$

between the previous and current frame, yielding an absolute camera pose $T_k \in \mathbb{R}^{4 \times 4}$ when we assume $T_1$ to be centered at the world origin. Our implementation uses a standard point-to-plane RGB-D registration implementation of Open3D [167].

Based on $F_k$ and $T_k$, we determine a per-pixel end-point-error $E_k$ between the estimated flow and the flow $\Psi_{T_k}$ we expect from a completely static scene where only the camera is moving by $T_k$, i.e.

$$E_k(u) = \| F_k(u) - \Psi_{T_k}(u) \|_2. \quad (2)$$

Using $D_k$, we can compute $\Psi_{T_k}(u)$ as the offset between $u$ and the corresponding point $u'$ projected from frame $k - 1$ into $k$. More specifically, let $\pi^{-1}$ be the backprojection operation of some fixed pinhole camera, such that $v = \pi^{-1}(u, D_k(u)) \in \mathbb{R}^3$ is the 3D coordinate of pixel $u$
with depth measurement $D_k(u)$ in the local coordinate system the camera, and let $\pi$ be the corresponding projection operation transforming $v$ back to $u$. $\Psi_k(u)$ is then given as

$$\Psi_k(u) = [\pi \circ T_{k-1} \circ T_k^{-1} \circ \pi^{-1}(u, D_k(u))] - u.$$  \hspace{1cm} (3)

Note that we imply proper conversion between euclidean and homogeneous coordinates by using the concatenation operator to keep the notation simple.

To decide which of the resulting scores $E_k(u)$ indicates dynamic regions, we analyze the histogram $H^i = (H^i_1, \ldots, H^i_n) \in \mathbb{N}^n$ of errors for each instance $i$. We chose the width of the $n$ histogram bins empirically as $c = 0.25$ based on the error values produced by our approach. As an indicator of the highest motion of $i$, we look for the rightmost mode $s_k(i) \in \mathbb{R}_{\geq 0}$ of $H^i$ that consists of at least $r = \sum_{l=1} H^i_l$ values, where $r \in [0, 1]$ is a hyperparameter. This means, we look for the bin index $j^\ast(i)$ with

$$j^\ast(i) = \max \left\{ j \mid H_{j-1}^i < H_j^i, H_{j+1}^i < H_j^i, \sum_{l=1}^j H_l^i \geq r \right\}$$

which, in turn, allows the mode $m_i$ to be defined as the center of histogram bin $j^\ast(i)$, i.e. $s_k(i) = (j^\ast(i) + 0.5)c$.

We normalize all scores by subtracting the smallest mode from them, assuming that at least one of the detections is of static nature. This is done to remove shifts in the overall error that can be caused by inaccurate estimates produced in $f_{\text{flow}}$ and $f_{\text{pose}}$. Together with an empirically chosen linear rescaling by a factor $\delta \in \mathbb{R}_{\geq 0}$, we get the normalized scores

$$E'_k(u) = \delta \cdot (E_k(u) - \min_i \{s_k(i)\})$$

that fulfill the previously mentioned criterion that scores $\leq 1$ are indicating a static object, while higher scores indicate dynamic regions.

While $E'_k(u)$ can now be used for the segmentation into static and dynamic regions, we found that the visualization of moving regions is more coherent if the segmentation happens on the object level. This is particularly important for articulated or non-rigid objects like humans, where potentially only a small part of the object (e.g. an arm) is moving. To accomplish this, we use the normalized modes $s'_k(i)$, which result from applying the transformation from (Equation (5)) to $m_i$. An instance $i$ is deemed as dynamic if $s'_k(i) \geq \tau$. To represent this in the resulting score map, we propagate this value in the final score map by setting $S_k(u) = s'_k(i)$ for all pixels $u$ with $\iota_k(u) = i$.

To make the dynamicity estimates more robust against noise in the error values when looking at multiple frames, we experimented with smoothing the values $s'_k(i)$ temporarily using the maximum over the current and a decaying previous score, such that the smoothed score of instance $i$ in frame $k$ is given as

$$s''_k(i) := \max\{\alpha \cdot s'_{k-1}(i), s'_k(i)\}.$$  \hspace{1cm} (6)

To make this work, we have to re-identify instance $i$ from frame $k-1$ in frame $k$. A priori, instance IDs do not have any relation to each other, because $f_{\text{seg}}$ is assumed to only be dependent on a single image. We use information about mask overlap between $\iota_k$, $L_k$ and $\iota'_{k-1}$, $L'_{k-1}$, where latter maps result from warping $\iota_k-1$, $L_k-1$ according to flow $F_k$, aligning them with the maps of frame $k$. A confusion matrix $C$ of the pairwise overlaps of the instance masks of the same class in $\iota_k$ and $\iota'_{k-1}$ is computed, such that

$$C_{ij} = \{u \mid \iota_k(u) = i, \iota'_{k-1}(u) = j, L_k(u) = L'_{k-1}(u)\}.$$  \hspace{1cm} (7)

We identify instance $i$ with instance $j'$ from the previous frame, if $j' = \arg\max_j \{C_{ij}\}$ and $C_{ij}$ is larger than a minimum overlap count. In addition, we also keep track of the average dynamicity scores over time to be able to give a sensible initial score estimate when detecting a new instance. A detailed explanation of this initialization scheme can be found in the supplemental material.

As the object tracking is only performed in 2D for efficiency reasons, we also accumulate the dynamicity scores of each instance over time in 2D by updating an accumulation map $A_k(u) \in \mathbb{R}_{\geq 0}$. To increase the interpretability of the scores, we compute a 3D end-point-error between last and current frame by using $F_k$ for the correspondences between the pixels and unprojecting the respective coordinates of into 3D using $\pi^{-1}$ with the corresponding depth maps and camera poses. The resulting 3D flow $\tilde{F}_k(u) \in \mathbb{R}^3$ is then combined with the warped previous accumulated score $A_{k-1}$ as $A_k(u) = A_{k-1}(u) + \|\tilde{F}_k(u)\|_2$.

### 3.2. Updating the Static Model

With the score map $S_k$ computed, we are able to integrate the static part of the frame into the static model. For this purpose, we use a modified version of real-time 3D reconstruction based on spatial voxel block hashing [95], where we added an extension for concurrent retrieval, insertion and removal of data [123]. However, in order to further increase the efficiency of our approach, we seamlessly shift potentially dynamic but currently static scene parts into the static scene representation until they become dynamic again. This requires us to additionally consider the following situations:

1. Dynamic regions should not be integrated into the static model. In case this happens erroneously, they should be removed as quickly as possible.

2. Regions that change their state from dynamic to static (e.g. a box is placed on a table) should be integrated into the static model seamlessly.

3. Regions changing their state from static to dynamic (e.g. a box is picked up) should be removed from the static model immediately.
4. Static regions that changed while not in the camera frustum should be updated as soon as new information is available.

Following the suggested modification of the weighting schema for dynamic object motion by Newcombe et al., we truncate the updated weight which effectively results in a moving average favoring newer measurements \[93\]. However, instead of having a global maximum weight \(W_{\eta} > 0\), we store a separate value \(W_{\eta,k}(v) > 0\) for each voxel \(v\) in our model and compute the new weight \(W_k(p) \in \mathbb{R}_{\geq 0}\) as

\[
W_k(v) = \min(W_{k-1}(v) + W_k'(v), W_{\eta,k}).
\]  

(8)

These maximum weights are computed directly from the dynamicity score. We found that a simple step function suffices, i.e.

\[
W_{\eta,k}(v) = \begin{cases} 
\hat{W}_\eta, & S_k(u_v) \leq \tau_\eta \\
\check{W}_\eta, & S_k(u_v) > \tau_\eta
\end{cases}
\]

(9)

given the corresponding raycasting source pixel \(u_v \in \mathcal{U}\) of voxel \(v\), a threshold \(\tau_\eta > 0\) and weight caps \(0 < \hat{W}_\eta \leq \check{W}_\eta\). This helps in situations 1 and 3, since dynamic regions are updated with new information more quickly, as well as in situation 4, as the weight is truncated even for static regions.

In addition, we aid the timely removal of dynamic regions from the static model (situations 1 and 3) by settings the SDF value to \(-1\) for voxels where the associated dynamicity score \(S_k(p)\) exceeds a threshold \(\tau_{SDF} > 0\). This, together with the high integration weight from before, invalidates the existing surface estimate at that location.

Situation 2 is already covered by the temporal smoothing of the dynamicity scores in Equation (6), because the decay parameter \(\alpha\) prevents to drop the scores too quickly, which leads to objects being considered dynamic for some time when they stop moving. Even though it takes a short time for the static reconstruction to integrate and stream the voxels of the state-changing object, we found this to be more intuitive when observing the live scene to have the object stop first than to suddenly disappear.

3.3. Visualization

After having streamed the hybrid scene representation to remote users’ devices, the static and dynamic scene entities have to be combined within an immersive scene exploration component, where we focus on virtual reality (VR) based immersion of users into the live-captured scenarios. For this, we created a client component that receives updates of the static model as well as the dynamic regions of the current RGB-D frame.

The static model is visualized as a mesh, where the local mesh representation of the static scene is updated using received MC voxel block indices and rendered in real-time, thereby following previous work \[123\]. In contrast, the dynamic parts are shown as a point cloud at the corresponding location relative to the static mesh. For this, we backproject the dynamic pixels of the current RGB-D frame using known camera intrinsics and the current camera pose.

The user is then able to individually and independently from the sensor explore the captured scene by physically looking and walking around or use a teleportation functionality for locomotion. The current position and orientation of the RGB-D sensor and other users is also shown.

3.4. Streaming

To be able to run the described method with low latency from the time of capturing to the visualization at remote locations, we use a server-client architecture. The server receives and distributes data packages over a network to the appropriate processing clients. The RGB-D capturing, segmentation into static and dynamic regions as well as the integration into the static model are performed in the reconstruction client.

Updates of this representation are then broadcasted to one or multiple exploration clients, which in turn update a mesh representation of the static scene using the MC indices. At the same time, the server also sends updates of the dynamic regions as masked RGB-D images together with the current camera pose estimate, such that the RGB-D pixels can be projected into the scene as a point-cloud.

3.5. Implementation Details

To take advantage of modern multi-processor architectures, the stages shown in Figure 2 are each running in separate processes, such that each stage can begin processing the next item once the current one has been processed. While this leads to overhead due to inter-process communication, the FPS of the pipeline is no longer bound to the latency, but the processing duration of the slowest stage in the pipeline. This can also be observed in our performance evaluation.

4. Experimental Results

To evaluate the performance of the proposed pipeline, we ran experiments on 8 self-recorded sequences captured with a Microsoft Azure Kinect RGB-D sensor in different office environments, and measured both speed and bandwidth metrics.

The scenes contain varying types of motion and we categorized them into three groups. Fixed (F.) are scenes that have no camera motion once dynamic entities can be seen in the camera, whereas Moving (M.) describes scenes with an always-moving camera and simultaneous object motion. A third category Outside (O.) contains a scene where the camera is hand-held, but object motion only happens outside of the camera view.
The bottom row of Figure 4 shows how the propagation of the error modes into the object masks aids to correctly identify potentially dynamic objects. Due to weak motion boundaries produced by \( f_{\text{flow}} \), a large region of pixels be-
Figure 3. Results of our approach on different scenes. Left to right: Input color image; resulting segmentation into static (blue) and dynamic (yellow) regions; the accumulated 3D flow magnitude; a novel view of the scene as visualized in the exploration client.

Figure 4. Comparison of design choices of the proposed pipeline. Top row: An example output from the exploration client using the standard voxelblock weighting schema (left) vs. exponential weight decay via weight capping. The second approach yields a reconstruction of the box with less artifacts. Bottom row: Thresholding of the normalized EPE before (left) and after (right) propagation of the error modes into the static (blue) and dynamic (yellow) object masks. Again, the second approach produces a more plausible segmentation into static and dynamic regions.

hind the moving person is considered dynamic after normalization. This can be filtered out completely in this case using our approach.

4.4. Limitations

While our approach shows promising results and is designed with modularity and extensibility in mind, there are also some limitations to consider. Most importantly, the pipeline only runs at interactive frame-rates due to the performance limitations inherited by the involved neural network approaches. In our scenario, we require high single-image inference speed, which is not a functionality modern deep learning approaches are particularly tuned for. Furthermore, our approach requires the segmentation network to detect objects to be able to identify dynamic regions, which limits its capabilities on out-of-distribution samples. This is also the case for the optical flow network, as it is also limited by the quality of the training data and the domain overlap with the scenes we recorded. In the supplemental material, we show some failure cases where failed detections of both $f_{seg}$ and $flow$ cause artifacts in the static reconstruction.

5. Conclusions

We presented a novel live-telepresence system that allows immersing remote users into live-captured environments with static and dynamic scene entities beyond room scale at practical bandwidth requirements. In order to allow the respectively required efficient 3D reconstruction, data streaming and VR-based visualization, we built our system upon a novel hybrid volumetric scene representation that combines a voxel-based representation of static scene geometry enriched by additional information regarding object semantics as well as their accumulated dynamic movement over time with a point-cloud-based representation for dynamic parts, where we perform the respective separation of static and dynamic parts based on optical flow and instance information extracted for the input frames. As a result of independently yet simultaneously streaming static and dynamic scene characteristics while keeping potentially moving but currently static scene entities in the static model as long as they remain static, as well as their fusion in the visualization on remote client hardware, we achieved VR-based live-telepresence in large-scale scenarios at interactive rates.

With the rapid improvements in hardware technology, particularly regarding GPUs, we expect our system to soon reach full real-time capability. Also, the modularity of our system allows replacing individual components with newer approaches, which might be particularly relevant for the instance segmentation network, which represents the main bottleneck of our current system.
Acknowledgements

This work was supported by the DFG project KL 1142/11-2 (DFG Research Unit FOR 2535 Anticipating Human Behavior).

References


[57] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? Advances in neural information processing systems, 30, 2017. 3


[72] Lingjie Liu, Marc Habermann, Viktor Rudnev, Kripasindhu Sarkar, Jiatao Gu, and Christian Theobalt. Neural actor: Neural free-view synthesis of human actors with pose con-


A. Detailed Explanation of Components

Filtering of Instance Proposals. When given a color image $I_k$, the instance segmentation network $f_{\text{seg}}$ used in our work outputs region proposals $M_1, ..., M_n$ in terms of Boolean masks that indicate the membership of each pixel $u \in \mathcal{U}$, i.e. $M_j(u) = 1$, if pixel $u$ belongs to proposal $j$, and $M_j(u) = 0$ otherwise. In addition, each mask is associated with a class label $l_j$ and a confidence score $c_j \in [0, 1]$. To produce the per-pixel class label $L_k$ and instance ID maps $i_k$, we have to integrate potentially overlapping region proposals, taking the confidence scores into account. We accomplish this by first removing proposals with a confidence smaller than a threshold $\tau_{\text{conf}}$. For each pixel $u$, we then find the instance ID of the proposal with maximum confidence, i.e. for the filtered indices $j_1', ..., j_n'$, we compute $i^*(u) = \arg\max_j \{c_j | M_j(u) = 1\}$. The resulting assignment is again filtered by removing IDs that do not exceed a minimum pixel count. In other words, we set

$$ \hat{i}(u) = \begin{cases} i^*(u), & \text{if } |\{i^*(u) = i\}| \geq \tau_{\text{count}} \\ 0, & \text{otherwise.} \end{cases} \quad (10) $$

As described in the paper, the resulting indices are then associated with the IDs from the previous frame to get the final map $i_k$ of instance IDs and the label map is set to the corresponding class labels $L_k(u) = l_{\hat{i}(u)}$.

Dynamicity Initialization. For instances that cannot be associated with an instance in the previous frame, we keep a map for the dynamicity scores of each class that is smoothed according to the scheme shown in Equation (6) of the main paper. More specifically, let $\tilde{s}_k : \mathcal{L} \to \mathbb{R}_{\geq 0}$ be defined as a mapping from the set of class labels $\mathcal{L}$ to the mean dynamicity scores, where

$$ \tilde{s}_k(l) := \max \left\{ \alpha_k \cdot \tilde{s}_{k-1}(l), \frac{1}{|\mathcal{L}|} \sum_{i \in \mathcal{L}} \tilde{s}_k(i) \right\}. \quad (11) $$

Figure 5 depicts the steps taken to compute the temporally smoothed score $s_k^*(i)$ of some instance $i$ with label $l_i$ from the initial score $s_k(i)$. Importantly, we also handle the case that a class is observed for the first time in the current frame. In that case, we set the class mean to the only available observation $s_k(i)$.

B. Hyperparameter Choices

The hyperparameters used for the performance evaluation and visualization were fixed for all scenes and are listed in Table 3.

C. Scene Descriptions

Table 5 contains a short description as well as some exemplary RGB images of each of the 8 self-recorded scenes used for our experiments. The scenes were all captured with a Microsoft Azure Kinect RGB-D sensor at a frame-rate of 30 Hz using the narrow FOV configuration and without depth binning.

D. Detailed Performance Comparison

In Table 6, we list the raw computation speed, measured in terms of frames per second (FPS), of individual components of our pipeline during evaluation. The results show that, on average, the inference of the instance segmentation network is the limiting factor for the overall speed of the pipeline. Note that the components run in parallel, such that actual processing speed of each component is limited by the output speed of the previous one. The measured values therefore only represent an upper bound for the FPS that each component can reach in our implementation. An example of this parallel execution is also visualized in Figure 7.

E. Comparison with Further Segmentation and Optical Flow Networks

To evaluate the choice of the instance segmentation [146] and optical flow [44] networks used in our approach, we compared the performance differences with some other recent segmentation [39,40,62,108] and optical flow [51,135,119].
### Table 3. Choices for the hyperparameters of the pipeline used during the evaluation. Symbol “N/A” indicates that no symbol was given to this parameter in the main publication or the supplementary material.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{\text{conf}}$</td>
<td>Minimum segmentation confidence</td>
<td>0.1</td>
</tr>
<tr>
<td>$\tau_{\text{count}}$</td>
<td>Minimum pixel count for class acceptance</td>
<td>2300</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Dynamic threshold</td>
<td>1.1</td>
</tr>
<tr>
<td>$\tau_{\text{V}}$</td>
<td>Voxel weight dynamicity threshold</td>
<td>1.1</td>
</tr>
<tr>
<td>$\tau_{\text{SDF}}$</td>
<td>SDF invalidation dynamicity threshold</td>
<td>1.1</td>
</tr>
<tr>
<td>$\hat{W}_{\eta}$</td>
<td>Static max. voxel weight</td>
<td>255</td>
</tr>
<tr>
<td>$\tilde{W}_{\eta}$</td>
<td>Dynamic max. voxel weight</td>
<td>3</td>
</tr>
<tr>
<td>$c$</td>
<td>Histogram bin width</td>
<td>0.25</td>
</tr>
<tr>
<td>N/A</td>
<td>Instance tracking min. overlap ratio</td>
<td>0.01</td>
</tr>
<tr>
<td>N/A</td>
<td>Instance dynamicity decay factor</td>
<td>0.2</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Class dynamicity decay factor</td>
<td>0.9</td>
</tr>
<tr>
<td>$\alpha_{\text{class}}$</td>
<td>Class dynamicity decay factor</td>
<td>0.9</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Dynamicity normalization scale factor</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 4. Performance comparison of our pipeline using different networks for instance segmentation and optical flow. For this purpose, we provide the resulting frame-rate (in FPS) our approach reaches using the given combinations of networks.

<table>
<thead>
<tr>
<th>Segmentation</th>
<th>Optical Flow</th>
<th>FPS [1/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask R-CNN [40]</td>
<td>LiteFlowNet2 [44]</td>
<td>6.91 (1.49)</td>
</tr>
<tr>
<td>PointRend [62]</td>
<td></td>
<td>5.34 (1.29)</td>
</tr>
<tr>
<td>SCNet [39]</td>
<td></td>
<td>4.17 (0.44)</td>
</tr>
<tr>
<td>DetectoRS [108]</td>
<td></td>
<td>2.39 (0.14)</td>
</tr>
<tr>
<td>SoloV2 [146]</td>
<td>MaskFlowNet [165]</td>
<td>9.57 (5.08)</td>
</tr>
<tr>
<td></td>
<td>RAFT [135]</td>
<td>2.88 (0.09)</td>
</tr>
<tr>
<td></td>
<td>GMA [51]</td>
<td>2.55 (0.08)</td>
</tr>
<tr>
<td>SoloV2 [146]</td>
<td>LiteFlowNet2 [44]</td>
<td>11.38 (5.51)</td>
</tr>
</tbody>
</table>

Table 4 shows our pipeline’s performance in terms of frames per second.

### F. Failure Cases

Our approach relies on accurate predictions from both the segmentation and optical flow networks. Objects not detected by the instance segmentation network affect their assignment to the static or dynamic scene parts, which is shown in Figure 6. Here, the balloon is not detected as an object by the SoloV2 network and is therefore erroneously integrated into the static model. While the integrated voxels conflict with future measurements and are eventually removed, they cause visually unpleasant artifacts during reconstruction. However, due to the modular nature of our approach, future developments with improved accuracy of the predictions might address this current limitation of our approach. Furthermore, future developments on increasing the efficiency of the networks for the respectively involved subtasks will further improve the overall performance.

Figure 6. Failure case of our method. Shown are RGB (top left), optical flow (top right), instance segmentation (bottom left) and resulting segmentation into static and dynamic (bottom right). Even though a clear motion cue is available in the optical flow image, due to a missing object detection, our method fails to correctly identify the dynamic region (orange circle).
<table>
<thead>
<tr>
<th>Scene</th>
<th>Description</th>
<th>Exemplary Images from Scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>items_1</td>
<td>A person moves around items (books and boxes) on an office table.</td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td>items_2</td>
<td>A person picks up and drops off items on a table in a medium-sized office.</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>people_1</td>
<td>Two persons meet at a coffee table and exchange a box.</td>
<td><img src="image3.png" alt="Image" /></td>
</tr>
<tr>
<td>people_2</td>
<td>Chairs and a boxes are moved around in an office seating area.</td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>people_3</td>
<td>Two persons exchange a small box.</td>
<td><img src="image5.png" alt="Image" /></td>
</tr>
<tr>
<td>ego_view</td>
<td>Balloons are kicked around in a medium-sized office.</td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td>oof_1</td>
<td>An office door is opened and closed while not seen by the camera.</td>
<td><img src="image7.png" alt="Image" /></td>
</tr>
<tr>
<td>oof_2</td>
<td>A box is moved multiple times while the camera is not observing it.</td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Table 5. Short description and exemplary images for each of the scenes used for the evaluation.
### Table 6

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>items_1</td>
<td>11.53</td>
<td>62.89</td>
<td>12.20</td>
<td>11.29</td>
<td>22.45</td>
<td>28.12</td>
<td>55.79</td>
<td>159.78</td>
<td>59.65</td>
</tr>
<tr>
<td>items_2</td>
<td>9.86</td>
<td>58.40</td>
<td>11.81</td>
<td>13.25</td>
<td>24.08</td>
<td>43.22</td>
<td>59.36</td>
<td>148.02</td>
<td>57.75</td>
</tr>
<tr>
<td>people_1</td>
<td>10.53</td>
<td>56.91</td>
<td>11.68</td>
<td>13.02</td>
<td>24.82</td>
<td>34.22</td>
<td>62.42</td>
<td>155.88</td>
<td>58.84</td>
</tr>
<tr>
<td>people_2</td>
<td>9.36</td>
<td>54.17</td>
<td>11.71</td>
<td>14.66</td>
<td>25.83</td>
<td>44.61</td>
<td>63.44</td>
<td>156.12</td>
<td>58.38</td>
</tr>
<tr>
<td>people_3</td>
<td>12.94</td>
<td>69.20</td>
<td>11.29</td>
<td>11.46</td>
<td>25.52</td>
<td>26.86</td>
<td>71.12</td>
<td>159.16</td>
<td>59.62</td>
</tr>
<tr>
<td>ego_view</td>
<td>9.73</td>
<td>51.88</td>
<td>11.92</td>
<td>16.29</td>
<td>25.76</td>
<td>49.17</td>
<td>56.43</td>
<td>156.67</td>
<td>59.23</td>
</tr>
<tr>
<td>oof_1</td>
<td>13.24</td>
<td>67.29</td>
<td>11.01</td>
<td>11.19</td>
<td>24.10</td>
<td>35.85</td>
<td>58.75</td>
<td>155.81</td>
<td>57.10</td>
</tr>
<tr>
<td>oof_2</td>
<td>10.63</td>
<td>56.02</td>
<td>11.87</td>
<td>14.99</td>
<td>25.45</td>
<td>41.48</td>
<td>59.37</td>
<td>154.40</td>
<td>58.75</td>
</tr>
<tr>
<td>Mean</td>
<td>10.98</td>
<td>59.59</td>
<td>11.56</td>
<td>13.27</td>
<td>24.65</td>
<td>37.08</td>
<td>60.84</td>
<td>155.73</td>
<td>58.67</td>
</tr>
</tbody>
</table>

Table 6. Raw computation speeds in FPS [1/s] for the major components of our pipeline, evaluated separately for each of the 8 scenes used for the evaluation. The last rows shows the mean over all scenes. The dynamicity computation is split into end-point-error computation (EPE), normalization (Norm.), temporal smoothing (Smooth.), object propagation (Prop.) and accumulation (Acc.).

![Gantt Chart](image_url)

Figure 7. Gantt chart showing the compute durations of the major components of our pipeline in a 700ms long section from the evaluation of scene items_1. The highlighted bars correspond to the same input frame. System components are optical flow estimation (Flow), instance segmentation (Seg.), odometry (Odom.), instance tracking (Track.), end-point-error computation (EPE), dynamicity normalization (Norm.), temporal smoothing (Smooth.), propagation (Prop.) and accumulation (Acc.). It can be seen that faster components of the pipeline have to wait for the next frame to become available to continue processing. Additional gaps result from scheduling and process communication.