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

Leif Van Holland[1] Patrick Stotko[1] Stefan Krumpen[1] Reinhard Klein[1] Michael Weinmann[1,2]

[1]University of Bonn [2]Delft University of Technology

In the following, we explain some steps from the main publication in more detail. Additionally, more information about the dataset and hyperparameters we used are given. Finally, a more in-depth performance analysis of different stages in the pipeline and a speed comparison of different methods for instance segmentation and optical flow estimation are shown.

A. Methodology

A.1. Segmentation Post-Processing

The raw output of the segmentation network f_{seg} may consist of multiple, potentially overlapping region proposals $M_1, ..., M_n$, which we integrate into the instance and label maps using non-maximum suppression. More specifically, the M_j are represented as Boolean masks that indicate the membership of each pixel, 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 ι_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 τ_{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 $j^*(u) = \text{argmax}_{j'}\{c_{j'} \mid M_{j'}(u) = 1\}$. The resulting assignment is again filtered by removing IDs that do not exceed a minimum pixel count in j^*. In other words, we set the instance ID map as

$$\iota_k(u) = \begin{cases} j^*(u), & \text{if } |\{i^*(u) = i\}| \geq \tau_{\text{count}} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The label map L_k is set to the corresponding class labels of the instance at each pixel, i.e., $L_k(u) = l_{\iota_k(u)}$. Note that the temporal smoothing step requires instance IDs of consecutive frames to be associated with one another. More details about this step can be found in Appendix A.5.

A.2. Optical Flow Weights

In order to filter out bad correspondences present in the output of NVIDIA's Optical Flow Accelerator (NVOFA) [8], we additionally compute the forward flow estimate $\bar{F}_k = f_{\text{flow}}(I_{k-1}, I_k)$. The estimator also provides per-pixel costs $C_k, \bar{C}_k$ for the estimates, which we combined with an error measure between forward and backward flow into a weight map W_k. For regions of small, smooth motion, the sum of forward and backward flow approximately cancels out and results in a vector field of small values. We therefore look at the norms of the vectors, $\|F_k + \bar{F}_k\|_2$, as an error between the two flows.

To get a single weight per pixel, we compute the geometric mean between the three terms

$$W_k = \sqrt[3]{(1 + \|F_k + \bar{F}_k\|_2)^{-1} (1 + \lambda C_k)^{-1} (1 + \lambda \bar{C}_k)^{-1}} \tag{4}$$

such that each term lies in the same range $[0, 1]$ and the result represents the level of confidence of the estimate at each pixel. $\lambda \in \mathbb{R}$ is a hyperparameter that scales the cost values of the estimator appropriately.

A.3. Expected Flow Computation

Using the current depth D_k, we can compute the expected flow $\Psi_k(u)$ as the offset between u and the corresponding point u' projected from frame $k - 1$ into k.

More specifically, let π^{-1} be the backprojection operation, such that $x = \pi^{-1}(u, D_k(u)) \in \mathbb{R}^3$ is the 3D coordinate of pixel u with depth measurement $D_k(u)$ in the local coordinate system of the camera, and let π be the corresponding projection operation transforming x back to u. The expected flow is therefore given as

$$\Psi_k(u) = \left[\pi \circ T_{k-1} \circ T_k^{-1} \circ \pi^{-1}(u, D_k(u))\right] - u. \tag{5}$$

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

A.4. Error Histogram Mode

To reduce the influence of the fluctuations of the average errors, we analyze the histogram $H = (H_1, ..., H_n) \in \mathbb{N}^n$ of the set of errors $\{E_k(u) \mid \iota_k(u) = i,\ u \in \mathcal{U}\}$ belonging to instance i. We chose the width c of the n histogram bins empirically as $c = 0.05$, based on the error values produced by our approach. H_j describes the number of pixels falling in bin j, which therefore corresponds to all pixels u of instance i where $j\,c \leq E_k(u) < (j+1)\,c$. The number of bins is chosen dynamically such that all error values are covered by a bin. 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 that consists of at least $r_{\text{mode}} \cdot \sum_l H_l$ values, where $r_{\text{mode}} \in [0, 1]$ is a hyperparameter. A bin j is a mode if the two adjacent bins have a smaller value, i.e., $H_{j-1} < H_j \wedge H_{j+1} < H_j$. This means, we look for the bin index $j^*(i)$ with

$$j^*(i) = \max \left\{ j \;\middle|\; H_j \text{ is a mode}, \frac{H_j}{\sum_l H_l} \geq r_{\text{mode}} \right\}, \quad (6)$$

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

A.5. Temporal Smoothing of Dynamicity Scores

To make the dynamicity estimates more robust against noise in the error values when looking at multiple frames, we experimented with smoothing the normalized rightmost error modes $s'_k(i)$ temporally 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

$$\hat{s}_k(i) := \max\{\alpha\, \hat{s}_{k-1}(i), s'_k(i)\}. \quad (7)$$

This maximum operation causes the smoothing to react to increasing dynamicity instantaneously, while slowly decaying if no increase in movement is detected. To compare the scores of instances in two consecutive frames, 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_{seg} is assumed to only be dependent on a single image. We use information about the mask overlap between ι_{k-1}, L_{k-1} and ι'_k, L'_k , where the latter maps result from warping ι_k, L_k according to the backward flow F_k, aligning them with the maps of frame $k-1$. A confusion matrix C of the pairwise overlaps of the instance masks of the same class in ι_{k-1} and ι'_k is computed, such that

$$C_{ij} = |\{u \mid \iota_{k-1}(u) = i, \iota'_k(u) = j, L_{k-1}(u) = L'_k(u)\}|. \quad (8)$$

We identify instance i with instance j' from the previous frame, if $j' = \text{argmax}_j\{C_{ij}\}$ and $C_{ij'}$ is larger than a minimum overlap ratio $r_{\text{track}} \in [0, 1]$. For instances that cannot be associated with an instance in the previous frame, we keep a mean for the dynamicity scores of each class that is smoothed according to the scheme shown in Equation (7). More specifically, let $\bar{s}_k : \mathcal{L} \to \mathbb{R}_{\geq 0}$ be defined as a mapping from the set of class labels $\mathcal{L}$ to the mean dynamicity scores for frame index k, where

$$\bar{s}_k(l) := \max \left\{ \alpha_{\text{class}}\, \bar{s}_{k-1}(l),\ \frac{1}{|\mathcal{I}_l|} \sum_{i \in \mathcal{I}_l} \hat{s}_k(i) \right\}. \quad (9)$$

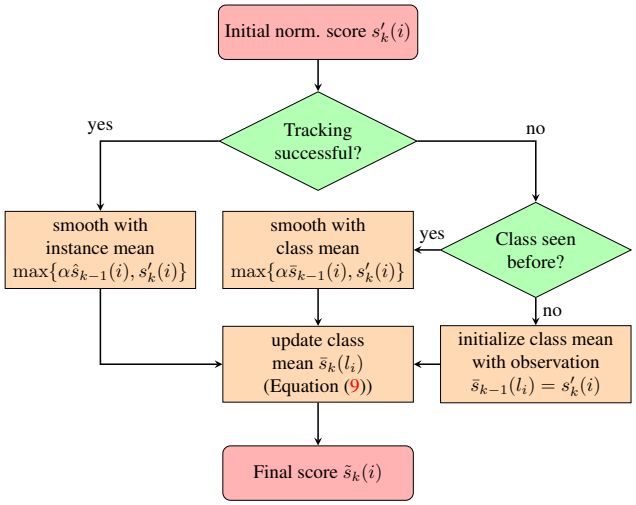


Figure 6: Flowchart visualizing the procedure for the temporal smoothing of the dynamicity score of instance i with class label l_i in frame k. If the tracking matches instance i with an instance from the previous frame, the resulting score is smoothed as shown in Equation (7). Otherwise, we first check whether the class was observed in a previous frame. In this case, we smooth the score similarly to Equation (7) but using the smoothed class mean from the last frame instead. In case the class is observed for the first time, we keep the initial score and use it as the new class mean for l_i. Lastly, the class mean for l_i is updated.

Figure 6 depicts the steps taken to compute the temporally smoothed score $\tilde{s}_k(i)$ of some instance i with label l_i from the initial normalized score $s'_k(i)$. Importantly, we also handle the case that a class is observed for the first time in the current frame, where we set the class mean to the only available observation $s'_k(i)$.

Note that this covers situation 2 (Section 3.2), because the decay parameter α prevents a sudden drop of the scores, 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. To ease this transition further, we added a small overlap period, in which objects in situation 2 are shown both as static and dynamic.

Symbol	Description	Value
τ_{conf}	Min. segmentation confidence	0.2
τ_{count}	Min. pixel count for class acceptance	2300
τ	Dynamic threshold	1.2
τ_η	Voxel weight dynamicity threshold	1.2
τ_{SDF}	SDF invalidation threshold	1.2
$\hat{W}_\eta$	Static max. voxel weight	255
$\check{W}_\eta$	Dynamic max. voxel weight	3
c	Histogram bin width	0.05
r_{mode}	Histogram relative min. bin count	0.01
r_{track}	Instance tracking min. overlap ratio	0.2
α	Instance dynamicity decay factor	0.99
α_{class}	Class dynamicity decay factor	0.99
δ	Dynamicity norm. scale factor	0.01
λ	NVOFA cost buffer scale factor	0.03

Table 3: Choices for the hyperparameters of the pipeline used for all sequences during the evaluation.

A.6. Moving Average of Voxel Weights

Instead of applying a global maximum voxel weight $W_\eta > 0$ in the integration step [7], we store a separate value $W_{\eta,k}(v) > 0$ for each voxel v in our model and compute the new weight $W_k(v) \in \mathbb{R}_{\geq 0}$ as

$$W_k(v) = \min(W_{k-1}(v) + W'_k(v), W_{\eta,k}). \tag{10}$$

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

$$W_{\eta,k}(v) = \begin{cases} \hat{W}_\eta, & S_k(u_v) \leq \tau_\eta \\ \check{W}_\eta, & S_k(u_v) > \tau_\eta \end{cases}, \tag{11}$$

given the corresponding ray casting source pixel $u_v \in \mathcal{U}$ of voxel v, a threshold $\tau_\eta > 0$ and weight caps $0 < \check{W}_\eta \leq \hat{W}_\eta$.

B. Experimental Results

A short description and some exemplary images of each scene we used for the evaluation of our method are shown in Table 6. Hyperparameters were fixed for all experiments and are listed in Table 3.

B.1. Detailed Performance Analysis

In Table 5, we list the raw computation speed of individual components of our pipeline during evaluation. Note that the components run in parallel, so the actual processing speed of each component is limited by the output speed of the previous one. The measured values therefore only represent a lower bound for the execution speed that each component can reach in our implementation. For example, for scene *items_1*, the segmentation inference component has the highest latency with 42.95 ms. All components can therefore process frames with a maximum frequency of $\frac{1000}{42.95\text{ ms}} \approx 23.28$ Hz. An example of this parallel execution is also visualized as a Gantt chart in Figure 7.

Segmentation	Optical Flow	FPS [1/s]
SCNet [1]	NVOFA [8]	4.47 (0.36)
PointRend [6]		6.60 (0.66)
Mask R-CNN [2]		8.04 (1.09)
SoloV2 [10]		14.95 (5.31)
YoloV8 [5]	GMA [4]	2.75 (0.11)
	RAFT [9]	3.02 (1.04)
	MaskFlowNet [11]	9.45 (1.17)
	LiteFlowNet2 [3]	13.69 (3.10)
YoloV8 [5]	NVOFA [8]	18.95 (5.75)

Table 4: Performance comparison of our pipeline using different approaches for instance segmentation and optical flow on the scene *items_1*. For this purpose, we provide the resulting frame-rate (standard deviation) our approach reaches using the given combinations of networks.

B.2. Comparison with Further Segmentation and Optical Flow Networks

To evaluate the choice of the instance segmentation network [5] and optical flow estimation [8] used in our approach, we compared the performance differences with some other recent segmentation [2, 1, 6, 10] and optical flow [9, 4, 11, 3] techniques. Table 4 shows our pipeline's performance in terms of frames per second when replacing either the segmentation or the optical flow approach. It can be seen that the combination we chose (bottom row) yields the highest processing speed.

Scene	System Components							
	Odometry	Optical Flow	Segmentation		Dynamicity			Voxel Hashing
			Inference	Tracking	EPE	Postproc.	Acc.	
items_1	24.55	28.40	42.95	15.00	26.60	21.75	25.75	33.45
items_2	21.80	28.80	44.70	13.60	25.70	19.80	25.10	38.60
people_1	24.10	28.70	42.45	14.90	25.80	19.80	25.55	34.05
people_2	24.80	29.20	44.70	15.90	26.60	21.00	26.00	43.30
people_3	25.20	29.60	45.10	16.40	26.90	20.80	26.70	46.10
people_4	22.55	29.05	46.20	15.75	26.10	20.85	26.20	41.35
people_5	25.10	28.53	41.97	14.47	26.10	19.80	25.30	40.90
ego_view	24.85	28.55	43.20	14.65	25.95	20.20	25.75	39.95
oof_1	24.30	28.90	38.25	12.20	25.15	18.85	24.75	41.00
oof_2	22.10	28.60	40.60	12.70	26.20	18.75	25.05	39.70
mean	23.94	28.83	43.01	14.56	26.11	20.16	25.62	39.84

Table 5: Raw computation speeds in milliseconds for the major components of our pipeline, evaluated separately for each of the 10 scenes used for the evaluation. The last row shows the mean over all scenes. The dynamicity computation is split into end-point-error computation (EPE), post-processing (Postproc., which includes normalization, temporal smoothing, and object propagation), accumulation (Acc.) and updating the static model (Voxel Hashing).

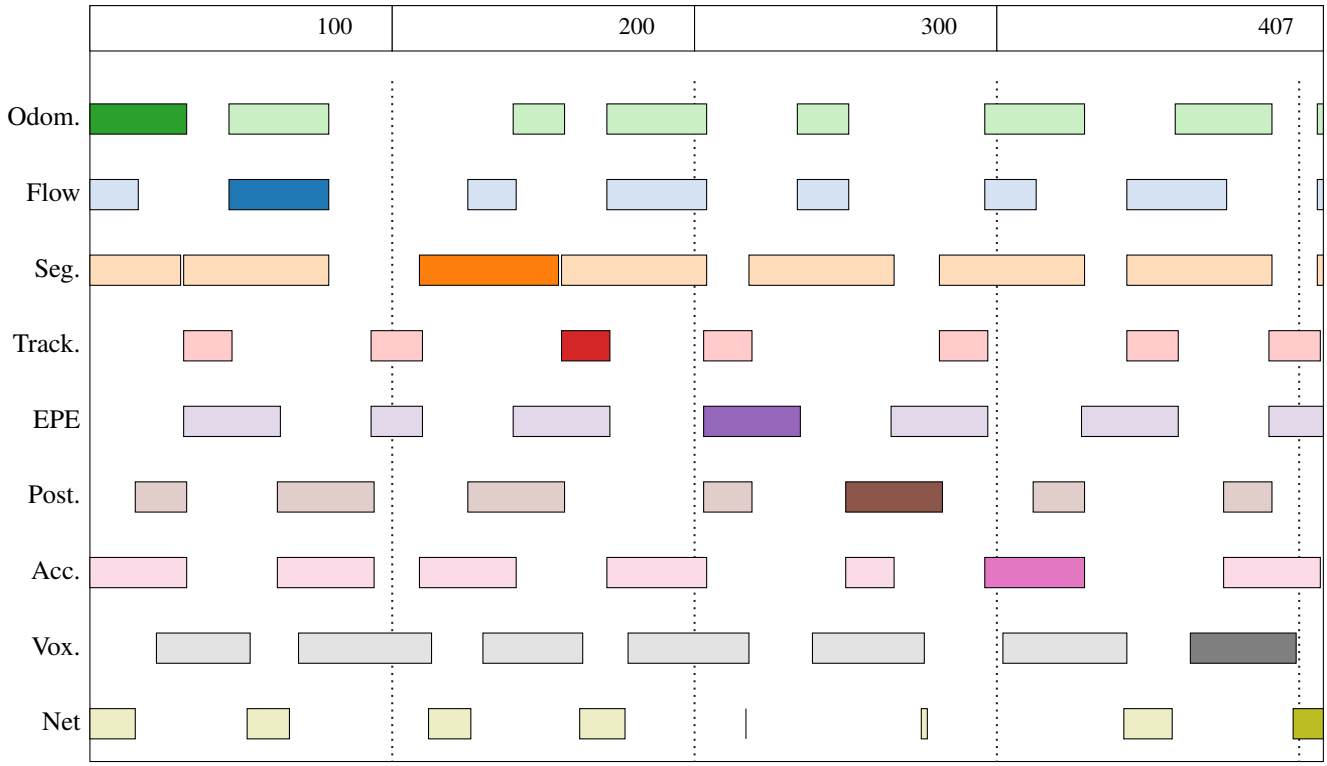


Figure 7: Gantt chart showing the compute durations of the major components of our pipeline on a single frame from arriving at the first process of the pipeline (0ms) to being received at the exploration client (407ms) from the evaluation of scene *items_1*. The highlighted bars correspond to the observed frame. System components are odometry (Odom.), optical flow estimation (Flow), instance segmentation (Seg.), instance tracking (Track.), end-point-error computation (EPE), dynamicity post-processing (Post.), dynamicity accumulation (Acc.), integration into the static model (Vox.) and network latency (Net.). 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.

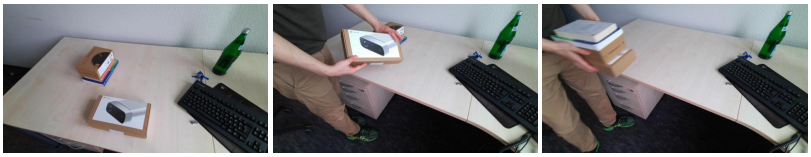
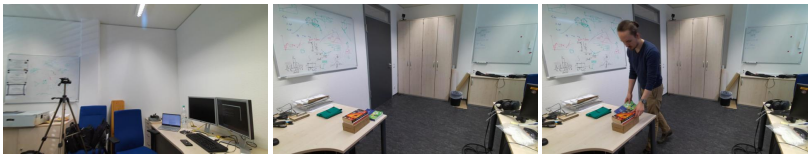
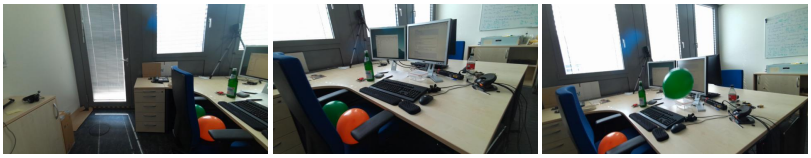
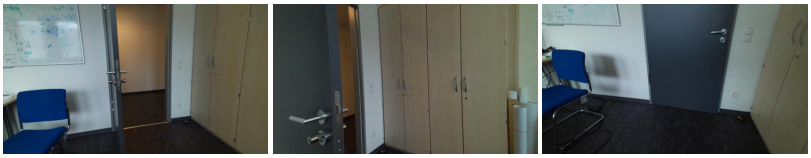
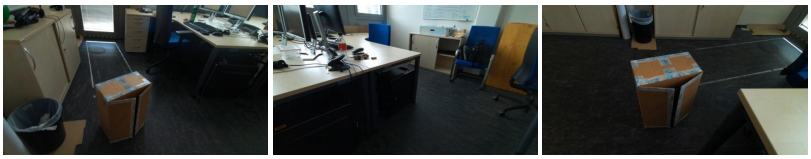
Scene	Description	Exemplary Images from Scene
items_1	A person moves around items (books and boxes) on an office table.	
items_2	A person picks up and drops off items on a table in a medium-sized office.	
people_1	Two persons meet at a coffee table and exchange a box.	
people_2	Chairs and a boxes are moved around in an office seating area.	
people_3	Two persons exchange a small box.	
people_4	Two persons meet in a library corner and exchange books.	
people_5	Two persons briefly meet and talk in a seating area.	
ego_view	Balloons are kicked around in a medium-sized office.	
oof_1	An office door is moved once while seen by the camera and once unseen.	
oof_2	A box is moved multiple times while the camera is not observing it.	

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

References

[1] Kai Han, Rafael S Rezende, Bumsub Ham, Kwan-Yee K Wong, Minsu Cho, Cordelia Schmid, and Jean Ponce. SCNet: Learning semantic correspondence. In *ICCV*. 2017. 3

[2] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*. 2017. 3

[3] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. A lightweight optical flow CNN—revisiting data fidelity and regularization. *TPAMI*, 2020. 3

[4] Shihao Jiang, Dylan Campbell, Yao Lu, Hongdong Li, and Richard Hartley. Learning to estimate hidden motions with global motion aggregation. In *ICCV*. 2021. 3

[5] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLOv8: The state-of-the-art YOLO model. https://github.com/ultralytics/ultralytics, 2023. Accessed: 2023-07-19. 3

[6] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. PointRend: Image segmentation as rendering. In *CVPR*. 2020. 3

[7] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, et al. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*. 2011. 3

[8] Nvidia Corporation. Nvidia optical flow sdk. https://developer.nvidia.com/opticalflow-sdk, 2019. Accessed: 2023-07-19. 1, 3

[9] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *ECCV*. 2020. 3

[10] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. SOLOv2: Dynamic and fast instance segmentation. *NeurIPS*, 2020. 3

[11] Shengyu Zhao, Yilun Sheng, Yue Dong, et al. MaskFlownet: Asymmetric feature matching with learnable occlusion mask. In *CVPR*. 2020. 3