Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion

Maik Keller
pmdtechnologies

Damien Lefloch
University of Siegen

Martin Lambers
University of Siegen

Shahram Izadi
Microsoft Research

Tim Weyrich
University College London

Andreas Kolb
University of Siegen

Abstract

Real-time or online 3D reconstruction has wide applicability and receives further interest due to availability of consumer depth cameras. Typical approaches use a moving sensor to accumulate depth measurements into a single model which is continuously refined. Designing such systems is an intricate balance between reconstruction quality, speed, spatial scale, and scene assumptions. Existing online methods either trade scale to achieve higher quality reconstructions of small objects/scenes. Or handle larger scenes by trading real-time performance and/or quality, or by limiting the bounds of the active reconstruction. Additionally, many systems assume a static scene, and cannot robustly handle scene motion or reconstructions that evolve to reflect scene changes. We address these limitations with a new system for real-time dense reconstruction with equivalent quality to existing online methods, but with support for additional spatial scale and robustness in dynamic scenes. Our system is designed around a simple and flat point-based representation, which directly works with the input acquired from range/depth sensors, without the overhead of converting between representations. The use of points enables speed and memory efficiency, directly leveraging the standard graphics pipeline for all central operations; i.e., camera pose estimation, data association, outlier removal, fusion of depth maps into a single denoised model, and detection and update of dynamic objects. We conclude with qualitative and quantitative results that highlight robust tracking and high quality reconstructions of a diverse set of scenes at varying scales.

1. Introduction and Background

Online 3D reconstruction receives much attention as inexpensive depth cameras (such as the Microsoft Kinect, Asus Xtion or PMD CamBoard) become widely available. Compared to offline 3D scanning approaches, the ability to obtain reconstructions in *real time* opens up a variety of interactive applications including: augmented reality (AR) where real-world geometry can be fused with 3D graphics and rendered live to the user; autonomous guidance for robots to reconstruct and respond rapidly to their environment; or even to provide immediate feedback to users during 3D scanning.

The first step of the reconstruction process is to acquire depth measurements either using sequences of regular 2D images (e.g. [19]), or with active sensors, such as laser scanners or depth cameras, based on triangulation or time-of-flight (ToF) techniques. Unlike methods that focus on reconstruction from a complete set of 3D points [5, 7], online methods require *fusion* of many overlapping depth maps into a single 3D representation that is continuously refined. Typically methods first find correspondences between depth maps (data association) and register or *align* depth maps to estimate the sensor's egomotion [1, 24]. The fusion method typically involves removal of outliers e.g. by visibility testing between depth maps [16], observing freespace violations [2], or photo-consistency [12], and merging of measurements into the global model, e.g. using simple weighted averaging [2] or more costly spatial regularization [25, 12].

Recent online systems [6, 11] achieve high-quality results by adopting the volumetric fusion method of Curless and Levoy [2]. This approach supports incremental updates, exploits redundant samples, makes no topological assumptions, approximates sensor uncertainty, and fusion is performed using a simple weighted average. For active sensors, this method produces very compelling results [2, 9, 6, 11]. The drawbacks are the *computational overheads* needed to continuously transition between different data representations: Where point-based input is converted to a continuous implicit function, discretized within a regular grid data structure, and converted back to an (explicit) form using expensive polygonization [10] or raycasting [14] methods. As well as the *memory overheads* imposed by using a regular voxel grid, which represents both empty space and surfaces densely, and thus greatly limits the size of the reconstruction volume.

These memory limitations have led to *moving-volume* systems [17, 23], which still operate on a very restricted volume, but free-up voxels as the sensor moves; or hierarchical volumetric data structures [26], which incur additional computational and data structure complexity for only limited gains in terms of spatial extent.

Beyond volumetric methods, simpler representations have

also been explored. Height-map representations [3] work with compact data structures allowing scalability, especially suited for modeling large buildings with floors and walls, since these appear as clear discontinuities in the height-map. Multi-layered height-maps support reconstruction of more complex 3D scenes such as balconies, doorways, and arches [3]. While these methods support compression of surface data for simple scenes, the 2.5D representation fails to model complex 3D environments efficiently.

Point-based representations are more amenable to the input acquired from depth/range sensors. [18] used a point-based method and custom structured light sensor to demonstrate in-hand online 3D scanning. Online model rendering required an intermediate volumetric data structure. Interestingly, an offline volumetric method [2] was used for higher quality final output, which nicely highlights the computational and quality trade-offs between point-based and volumetric methods. [22] took this one step further, demonstrating higher quality scanning of small objects using a higher resolution custom structured light camera, sensor drift correction, and higher quality surfel-based [15] rendering. These systems however focus on single small object scanning. Further, the sensors produce less noise than consumer depth cameras (due to dynamic rather than fixed structured light patterns), making model denoising less challenging.

Beyond reducing computational complexity, point-based methods lower the memory overhead associated with volumetric (regular grid) approaches, as long as overlapping points are merged. Such methods have therefore been used in larger sized reconstructions [4, 20]. However, a clear trade-off becomes apparent in terms of scale versus speed and quality. For example, [4] allow for reconstructions of entire floors of a building (with support for loop closure and bundle adjustment), but frame rate is limited ($\sim$ 3 Hz) and an unoptimized surfel map representation for merging 3D points can take seconds to compute. [20] use a multi-level surfel representation that achieves interactive rates ($\sim$ 10 Hz) but require an intermediate octree representation which limits scalability and adds computational complexity.

In this paper we present an online reconstruction system also based around a flat, point-based representation, rather than any spatial data structure. A key contribution is that our system is memory-efficient, supporting spatially extended reconstructions, but without trading reconstruction quality or frame rate. As we will show, the ability to directly render the representation using the standard graphics pipeline, without converting between multiple representations, enables efficient implementation of all central operations, i.e., camera pose estimation, data association, denoising and fusion through data accumulation, and outlier removal.

A core technical contribution is leveraging a fusion method that closely resembles [2] but removes the voxel grid all-together. Despite the lack of a spatial data structure, our system still captures many benefits of volumetric fusion, with competitive performance and quality to previous online systems, allowing for accumulation of denoised 3D models over time that exploit redundant samples, model measurement uncertainty, and make no topological assumptions.

The simplicity of our approach allows us to tackle another fundamental challenge of online reconstruction systems: the assumption of a static scene. Most previous systems make this assumption or treat dynamic content as outliers [18, 22]; only KinectFusion [6] is at least capable of reconstructing moving objects in a scene, provided a static pre-scan of the background is first acquired. Instead, we leverage the immediacy of our representation to design a method to not only robustly segment dynamic objects in the scene, which greatly improves the robustness of the camera pose estimation, but also to continuously update the global reconstruction, regardless of whether objects are added or removed. Our approach is further able to detect when a moving object has become static or a stationary object has become dynamic.

The ability to support reconstructions at quality comparable to state-of-the-art, without trading real-time performance, with the addition of extended spatial scale and support for dynamic scenes provides unique capabilities over prior work. We conclude with results from reconstructing a variety of static and dynamic scenes of different scales, and an experimental comparison to related systems.

2. System Overview

Our high-level approach shares commonalities with the existing incremental reconstruction systems (presented previously): we use samples from a moving depth sensor; first preprocess the depth data; then estimate the current six degree-of-freedom (6DoF) pose of sensor relative to the scene; and finally use this pose to convert depth samples into a unified coordinate space and fuse them into an accumulated global model. Unlike prior systems, we adopt a purely point-based representation throughout our pipeline, carefully designed to support data fusion with quality comparable to online volumetric methods, whilst enabling real-time reconstructions at extended scales and in dynamic scenes.

Our choice of representation makes our pipeline extremely amenable to implementation using commodity graphics hardware. The main system pipeline as shown in Fig. 1 is based on the following steps:

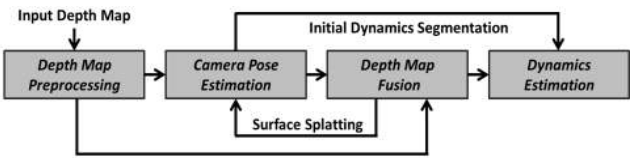

Figure 1. Main system pipeline.

Depth Map Preprocessing Using the intrinsic parameters of the camera, each input depth map from the depth

sensor is transformed into a set of 3D points, stored in a 2D *vertex map*. Corresponding normals are computed from central-differences of the denoised vertex positions, and per-point radii are computed as a function of depth and gradient (stored in respective *normal* and *radius* maps).

Depth Map Fusion Given a valid camera pose, input points are fused into the *global model*. The global model is simply a list of 3D points with associated attributes. Points evolve from *unstable* to *stable* status based on the confidence they gathered (essentially a function of how often they are observed by the sensor). Data fusion first *projectively associates* each point in the input depth map with the set of points in the global model, by rendering the model as an *index map*. If corresponding points are found, the most reliable point is merged with the new point estimate using a weighted average. If no reliable corresponding points are found, the new point estimate is added to the global model as an unstable point. The global model is cleaned up over time to remove outliers due to visibility and temporal constraints. Sec. 4 discusses our point-based data fusion in detail.

Camera Pose Estimation All established (high confidence) model points are passed to the visualization stage, which reconstructs dense surfaces using a surface splatting technique (see Sec. 5). To estimate the 6DoF camera pose, the model points are projected from the previous camera pose, and a pyramid-based dense iterative closest point (ICP) [11] alignment is performed using this rendered *model map* and input depth map. This provides a new relative rigid 6DoF transformation that maps from the previous to new global camera pose. Pose estimation occurs prior to data fusion, to ensure the correct projection during data association.

Dynamics Estimation A key feature of our method is automatic detection of dynamic changes in the scene, to update the global reconstruction and support robust camera tracking. Dynamic objects are initially indicated by outliers in point correspondences during ICP. Starting from these areas, we perform a point-based region growing procedure to identify dynamic regions. These regions are excluded from the camera pose estimate, and their corresponding points in the global model are reset to *unstable* status, leading to a natural propagation of scene changes into our depth map fusion. For more detail, see Sec. 6.

3. Depth Map Preprocessing

We denote a 2D pixel as $\boldsymbol{u} = (x,y)^\top \in \mathbb{R}^2$. $\mathcal{D}_i \in \mathbb{R}$ is the raw depth map at time frame i. Given the intrinsic camera calibration matrix $\boldsymbol{K}_i$, we transform $\mathcal{D}_i$ into a corresponding vertex map $\mathcal{V}_i$, by converting each depth sample $\mathcal{D}_i(\boldsymbol{u})$ into a vertex position $\boldsymbol{v}_i(\boldsymbol{u}) = \mathcal{D}_i(\boldsymbol{u})\boldsymbol{K}_i^{-1}(\boldsymbol{u}^\top,1)^\top \in \mathbb{R}^3$ in camera space. A corresponding normal map $\mathcal{N}_i$ is determined from central-differences of the vertex map. A copy of the depth map (and hence associated vertices and normals) are also denoised using a *bilateral filter* [21] (for camera pose estimation later).

The 6DoF camera pose transformation comprises of rotation ($\mathrm{R}_i \in \mathbb{SO}_3$) matrix and translation ($\boldsymbol{t}_i \in \mathbb{R}^3$) vector, computed per frame i as $\boldsymbol{T}_i = [\mathrm{R}_i, \boldsymbol{t}_i] \in \mathbb{SE}_3$. A vertex is converted to global coordinates as $\boldsymbol{v}_i^{\mathrm{g}} = \boldsymbol{T}_i \boldsymbol{v}_i$. The associated normal is converted to global coordinates as $\boldsymbol{n}_i^{\mathrm{g}}(\boldsymbol{u}) = \mathrm{R}_i \boldsymbol{n}_i(\boldsymbol{u})$. Multi-scale pyramids $\mathcal{V}_i^l$ and $\mathcal{N}_i^l$ are computed from vertex and normal maps for hierarchical ICP, where $l \in \{0,1,2\}$ and $l = 0$ denotes the original input resolution (e.g. 640×480 for Kinect or 200×200 for PMD CamBoard).

Each input vertex also has an associated radius $r_i(\boldsymbol{u}) \in \mathbb{R}$ (collectively stored in a radius map $\mathcal{R}_i \in \mathbb{R}$), determined as in [22]. To prevent arbitrarily large radii from oblique views, we clamp radii for grazing observations exceeding 75°.

In the remainder, we omit time frame indices i for clarity, unless we refer to two different time frames at once.

4. Depth Map Fusion

Our system maintains a single global model, which is simply an unstructured set of points $\bar{P}_k$ each with associated position $\bar{\boldsymbol{v}}_k \in \mathbb{R}^3$, normal $\bar{\boldsymbol{n}}_k \in \mathbb{R}^3$, radius $\bar{r}_k \in \mathbb{R}$, confidence counter $\bar{c}_k \in \mathbb{R}$, and time stamp $\bar{t}_k \in \mathbb{N}$, stored in a flat array indexed by $k \in \mathbb{N}$.

New measurements $\boldsymbol{v}$ are either added as or merged with unstable points, or they get merged with stable model points. Merging $\boldsymbol{v}$ with a point $\bar{P}_k$ in the global model increases the confidence counter $\bar{c}_k$. Eventually an unstable point changes its status to stable: points with $\bar{c}_k \geq c_{\mathrm{stable}}$ are considered stable (in practice $c_{\mathrm{stable}} = 10$). In specific temporal or geometric conditions, points are removed from the global model.

4.1. Data Association

After estimation of the camera pose of the current input frame (see Sec. 5), each vertex $\boldsymbol{v}^{\mathrm{g}}$ and associated normal and radius are integrated into the global model.

In a first step, for each valid vertex $\boldsymbol{v}^{\mathrm{g}}$, we find potential corresponding points on the global model. Given the inverse global camera pose $\boldsymbol{T}^{-1}$ and intrinsics $\boldsymbol{K}$, each point $\bar{P}_k$ in the global model can be projected onto the image plane of the current physical camera view, where the respective point index k is stored: we render all model points into a sparse *index map* $\mathcal{I}$. Unlike the splat-based dense surface reconstruction renderer used in other parts of our pipeline (see Sec. 5), this stage renders each point index into a single pixel to reveal the actual surface sample distribution.

As nearby model points may project onto the same pixel, we increase the precision of $\mathcal{I}$ by supersampling, representing $\mathcal{I}$ at 4×4 the resolution of the input depth map. We start identifying model points near $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$ by collecting point indices within the 4×4-neighborhood around each input pixel location $\boldsymbol{u}$ (suitably coordinate-transformed from $\mathcal{D}$ to $\mathcal{I}$).

Amongst those points, we determine a single corresponding model point by applying the following criteria:

1. Discard points larger than $\pm\delta_{\text{depth}}$ distance from the viewing ray $\boldsymbol{v}^{\text{g}}(\boldsymbol{u})$ (the sensor line of sight), with δ_{depth} adapted according to sensor uncertainty (i.e. as a function of depth for triangulation-based methods [13]).
2. Discard points whose normals have an angle larger than δ_{norm} to the normal $\boldsymbol{n}^{\text{g}}(\boldsymbol{u})$. (We use $\delta_{\text{norm}} = 20^{\circ}$.)
3. From the remaining points, select the ones with the highest confidence count.
4. If multiple such points exist, select the one closest to the viewing ray through $\boldsymbol{v}^{\text{g}}(\boldsymbol{u})$.

4.2. Point Averaging with Sensor Uncertainty

If a corresponding model point $\bar{P}_k$ is found during data association, this is averaged with the input vertex $\boldsymbol{v}^{\text{g}}(\boldsymbol{u})$ and normal $\boldsymbol{n}^{\text{g}}(\boldsymbol{u})$ as follows:

$$\bar{\boldsymbol{v}}_k \leftarrow \frac{\bar{c}_k \bar{\boldsymbol{v}}_k + \alpha \boldsymbol{v}^{\text{g}}(\boldsymbol{u})}{\bar{c}_k + \alpha}, \quad \bar{\boldsymbol{n}}_k \leftarrow \frac{\bar{c}_k \bar{\boldsymbol{n}}_k + \alpha \boldsymbol{n}^{\text{g}}(\boldsymbol{u})}{\bar{c}_k + \alpha}, \tag{1}$$

$$\bar{c}_k \leftarrow \bar{c}_k + \alpha, \quad \bar{t}_k \leftarrow t, \tag{2}$$

where t is a new time stamp. Our weighted average is distinct from the original KinectFusion system [11], as we introduce an explicit sample confidence α. This applies a Gaussian weight on the current depth measurement as $\alpha = \mathrm{e}^{-\gamma^2/2\sigma^2}$, where γ is the normalized radial distance of the current depth measurement from the camera center, and $\sigma = 0.6$ is derived empirically. This approach weights measurements based on the assumption that measurements closer to the sensor center will increase in accuracy [2]. As shown in Fig. 2, modeling this sensor uncertainty leads to higher quality denoising.

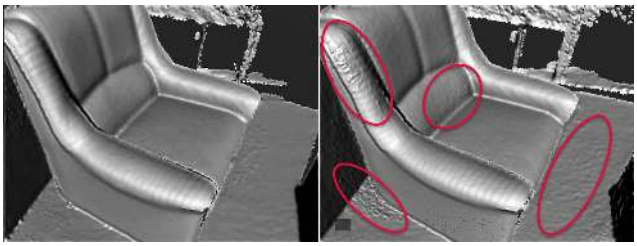

Figure 2. Weighted averaging of points using our method (left) and the method of [11] (right).

Since the noise level of the input measurement increases as a function of depth [13], we apply Eqs. (1) only if the radius of the new point is not significantly larger than the radius of the model point, i.e., if $r(\boldsymbol{u}) \leq (1 + \delta_r)\bar{r}$; we empirically chose $\delta_r = 1/2$. This ensures that we always refine details, but never coarsen the global model. We apply the time stamp and the confidence counter updates according to Eqs. (2) irrespectively.

If no corresponding model point has been identified, a new unstable point is added to the global model with $\bar{c}_k = \alpha$, containing the input vertex, normal and radius.

4.3. Removing Points

So far we have merged or added new measurements to the global model. Another key step is to remove points from our global model due to various conditions:

1. Points that remain in the unstable state for a long time are likely outliers or artifacts from moving objects and will be removed after t_{max} time steps.
2. For stable model points that are merged with new data, we remove all model points that lie in front of these newly merged points, as these are free-space violations. To find these points to remove, we use the index map again and search the neighborhood around the pixel location that the merged point projects onto[1]. This is similar in spirit to the free-space carving method of [2], but avoids expensive voxel space traversal.
3. If after averaging, a stable point has neighboring points (identified again via the index map) with very similar position and normal and their radii overlap, then we merge these redundant neighboring points to further simplify the model.

Points are first marked to be removed from $\bar{P}_k$, and in a second pass, the list is sorted (using a fast radix sort implementation), moving all marked points to the end, and finally items deleted.

5. Camera Pose Estimation

Following the approach of KinectFusion [11], our camera pose estimation uses dense hierarchical ICP to align the bilateral filtered input depth map $\mathcal{D}_i$ (of the current frame i) with the reconstructed model by rendering the model into a virtual depth map, or *model map*, $\hat{\mathcal{D}}_{i-1}$, as seen from the previous frame's camera pose $\boldsymbol{T}_{i-1}$. We use 3 hierarchy levels, with the finest level at the camera's resolution; unstable model points are ignored. The registration transformation provides the relative change from $\boldsymbol{T}_{i-1}$ to $\boldsymbol{T}_i$.

While KinectFusion employs raycasting of the (implicit) voxel-based reconstruction, we render our explicit, point-based representation using a simple surface-splatting technique: we render overlapping, disk-shaped *surface splats* that are spanned by the model point's position $\bar{\boldsymbol{v}}$, radius $\bar{r}$ and orientation $\bar{\boldsymbol{n}}$. Unlike more refined surface-splatting techniques, such as EWA Surface Splatting [27], we do not perform blending and analytical prefiltering of splats but trade local surface reconstruction quality for performance by simply rendering opaque splats.

We use the same point-based renderer for user feedback, but add Phong shading of surface splats, and also overlay the dynamic regions of the input depth map.

[1] Backfacing points that are close to the merged points remain protected—such points may occur in regions of high curvature or around thin geometry in the presence of noise and slight registration errors. Furthermore, we protect points that would be consistent with direct neighbor pixels in $\mathcal{D}$, to avoid spurious removal of points around depth discontinuities.

6. Dynamics Estimation

The system as described above already has limited support for dynamic objects, in that unstable points must gain confidence to be promoted to stable model points, and so fast moving objects will be added and then deleted from the global model. In this section we describe additional steps that lead to an explicit classification of observed points as being part of a dynamic object. In addition, we aim at segmenting entire objects whose surface is partially moving and remove them from the global point model.

We build upon an observation by Izadi et al. [6]: when performing ICP, failure of data association to find model correspondences for input points is a strong indication that these points are depth samples belonging to dynamic objects. Accordingly, we retrieve this information by constructing an ICP status map $\mathcal{S}$ (with elements $s_i(\boldsymbol{u})$) that encodes for each depth sample the return state of ICP's search for a corresponding model point in the data association step:

`no_input`:	$\boldsymbol{v}_k(\boldsymbol{u})$ is invalid or missing.
`no_cand`:	No stable model points in proximity of $\boldsymbol{v}_k(\boldsymbol{u})$.
`no_corr`:	Stable model points in proximity of, but no valid ICP correspondence for $\boldsymbol{v}_k(\boldsymbol{u})$.
`corr`:	Otherwise ICP found a correspondence.

Input points marked as `no_corr` are a strong initial estimate of parts of the scene that move independent of camera motion, i.e. dynamic objects in the scene.

We use these points to seed our segmentation method based on hierarchical region growing (see below). It creates a *dynamics map* $\mathcal{X}$, storing flags $x_i(\boldsymbol{u})$, that segments the current input frame into static and dynamic points. The region growing aims at marking complete objects as dynamic even if only parts of them actually move. (Note that this high-level view on dynamics is an improvement over the limited handling of dynamics in previous approaches, e.g., [6].)

In the depth map fusion stage, model points that are merged with input points marked as dynamic are potentially demoted to unstable points using the following rule:

$$\text{if} \quad x_i(\boldsymbol{u}) \wedge \bar{c}_k \geq c_{\text{stable}} + 1 \quad \text{then} \quad \bar{c}_k \leftarrow 1 \qquad (3)$$

Thus, the state change from static to dynamic is reflected immediately in the model. A critical aspect is the offset of $+1$ in Eq. (3): it ensures that any dynamic point that sufficiently grew in confidence (potentially because it is now static) is allowed to be added to the global model for at least one iteration; otherwise, a surface that has once been classified as dynamic would never be able to re-added to the global model, as it would always be inconsistent with the model, leading to `no_corr` classification.

For the bulk of the time, however, dynamic points remain *unstable* and as such are not considered for camera pose estimation (see Sec. 5), which greatly improves accuracy and robustness of $\boldsymbol{T}$.

Hierarchical Region Growing The remainder of this section explains the region growing-based segmentation approach that computes the map $\mathcal{X}$.

The goal is essentially to find connected components in $\mathcal{D}$. In the absence of explicit neighborhood relations in the point data, we perform region growing based on point attribute similarity. Starting from the seed points marked in $\mathcal{X}$, we agglomerate points whose position and normal are within given thresholds of vertex $\boldsymbol{v}(\boldsymbol{u})$ and normal $\boldsymbol{n}(\boldsymbol{u})$ of a neighbor with $x(\boldsymbol{u}) = \text{true}$.

To accelerate the process, we start at a downsampled $\mathcal{X}^2$, and repeatedly upsample until we reach $\mathcal{X}^0 = \mathcal{X}$, each time resuming region growing. (We reuse the input pyramids built for camera pose estimation.)

We improve robustness to camera noise and occlusions by removing stray `no_corr` points through morphological erosion at the coarsest pyramid level $\mathcal{X}^2$ after initializing it from $\mathcal{S}$. This also ensures that $\mathcal{X}^2$ covers only the inner region of dynamic objects.

7. Results

We have tested our system on a variety of scenes (see Table 1). Fig. 3 shows a synthetic scene *Sim*. We generated rendered depth maps for a virtual camera rotating around

	#frames input/processed (fps in./proc.)	#model-points	Avg. timings [ms] ICP	Dyn-Seg.	Fusion
Sim	950/950 (15/15)	467,200	18.90	2.03	11.50
Flower-pot	600/480 (30/24)	496,260	15.87	1.90	6.89
Teapot	1000/923 (30/27)	191,459	15.20	1.60	5.56
Large Office	11892/6704 (30/17)	4,610,800	21.75	2.39	13.90
Moving Person	912/623 (30/20)	210,500	15.92	3.23	16.61
Ball-game	1886/1273 (30/21)	350,940	16.74	3.15	17.66
PMD	4101/4101 (27/27)	280,050	10.70	0.73	3.06

Table 1. Results from test scenes obtained on a PC equipped with an Intel i7 8-core CPU and an NVidia GTX 680 GPU. Input frames have a size of 640×480 pixels, except for the *PMD* scene which uses a frame size of 200×200.

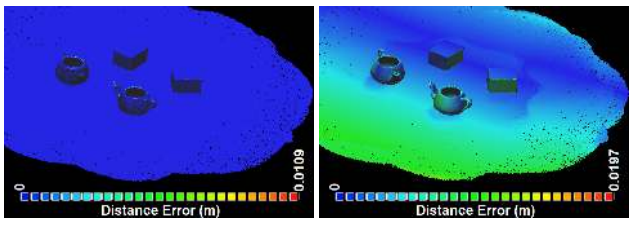

Figure 3. The synthetic scene *Sim*. Left: error in final global model based on ground truth camera transformations. Right: final error based on ICP pose estimation[2].

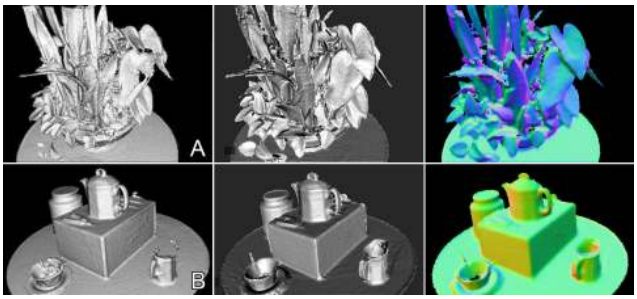

Figure 4. The scenes *Flowerpot* (top row) and *Teapot* (bottom row). A and B show reconstruction results of the original KinectFusion system. The other images show our method (middle: phong-shaded surfels, right: model points colored with surface normals).

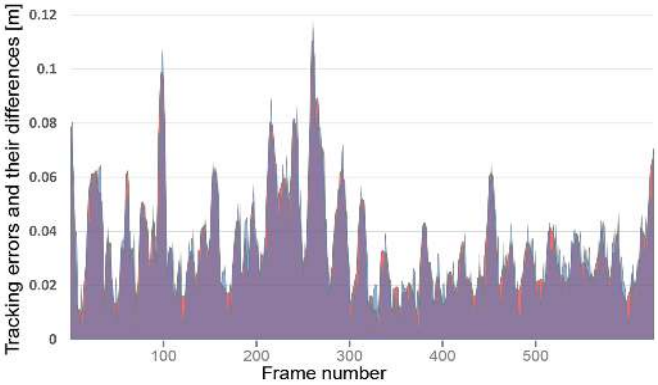

Figure 5. Tracking errors for the original KinectFusion system compared to our point-based approach. Tracking results were computed on the *Flowerpot* sequence, by subtracting Vicon ground truth data from the resulting per frame 3D camera position. For each system, error is computed as the absolute distance between the estimated camera position and the ground truth position (after aligning both coordinate spaces manually). Where the error of the original KinectFusion exceeds that of the new, the gap is colored blue. Where the error of our method exceeds the original, the gap is colored red. Note our method is similar in performance with the largest delta being $\sim 1cm$.

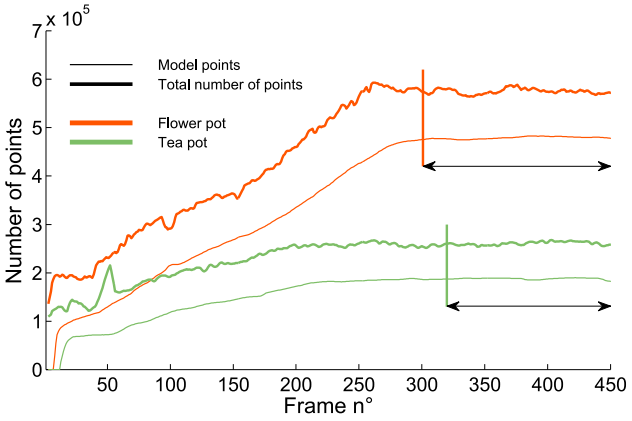

Figure 6. The number of global model points stored on the GPU plotted over time for the *Flowerpot* and *Teapot* scenes. Note after the completion of one full turn of the turntable, the number of points converges instead of continuously growing.

this scene and used these as input to our system. This gave us ground truth camera transformations $\boldsymbol{T}_i^{\mathrm{GT}}$ and ground truth scene geometry. Using $\boldsymbol{T}_i^{\mathrm{GT}}$, the points in the resulting global model have a mean position error of 0.019 mm. This demonstrates only minimal error for our point-based data fusion approach. The camera transformations $\boldsymbol{T}_i$ obtained from ICP have a mean position error of 0.87 cm and a mean viewing direction error of 0.1 degrees. This results in a mean position error of 0.20 cm for global model points.

The *Flowerpot* and *Teapot* scenes shown in Fig. 4 were recorded by Nguyen et al. [13]. Objects are placed on a turntable which is rotated around a stationary Kinect camera. Vicon is used for ground truth pose estimation of the Kinect, which are compared to ICP for our method and the original KinectFusion system Fig. 5

Fig. 6 shows that the number of global model points for these scenes remains roughly constant after one full turn of the turntable. This demonstrates that new points are not continuously added; and the global model is refined but kept compact. Note that one Kinect camera input frame provides up to 307,200 input points, but the total number of points in the final global teapot model is less than 300,000.

The *Large Office* scene shown in Fig. 7 consists of two rooms with a total spatial extent of approximately $10m \times 6m \times 2.5m$. A predefined volumetric grid with 32-bit voxels and 512 MB of GPU memory would result in a voxel size of more than 1 cm^3. In contrast, our system does not define the scene extents in advance: the global model grows as required. Furthermore, it does not limit the size of representable details; Fig. 7 shows close-ups of details on the millimeter scale (e.g. the telephone keys). The 4.6 million global model points reported in Tab. 1 can be stored in 110 MB of GPU memory using 3 floating point values for the point position, 2 for the normalized point normal, 1 for the radius, and one extra byte for a confidence counter. Additionally, RGB colors can be stored for each global point, to texture the final model (see Fig. 7 far right). Rather than merge RGB samples, we simply store the last one currently.

In the *Moving Person* scene shown in Fig. 8, the person first sits in front of the sensor and is reconstructed before moving out of view. Since the moving person occupies much of the field of view, leaving only few reliable points for ICP, camera tracking fails with previous approaches (see e.g. Izadi et al. Fig. 8 [6]). Our system segments the moving person and ignores dynamic scene parts in the ICP stage, thereby ensuring robustness to dynamic motion.

The *Ballgame* scene shown in Fig. 9 shows two people playing with a ball across a table. Our region growing approach segments dynamics on the object level instead of just

[2]Rendered using CloudCompare, http://www.danielgm.net/cc/.

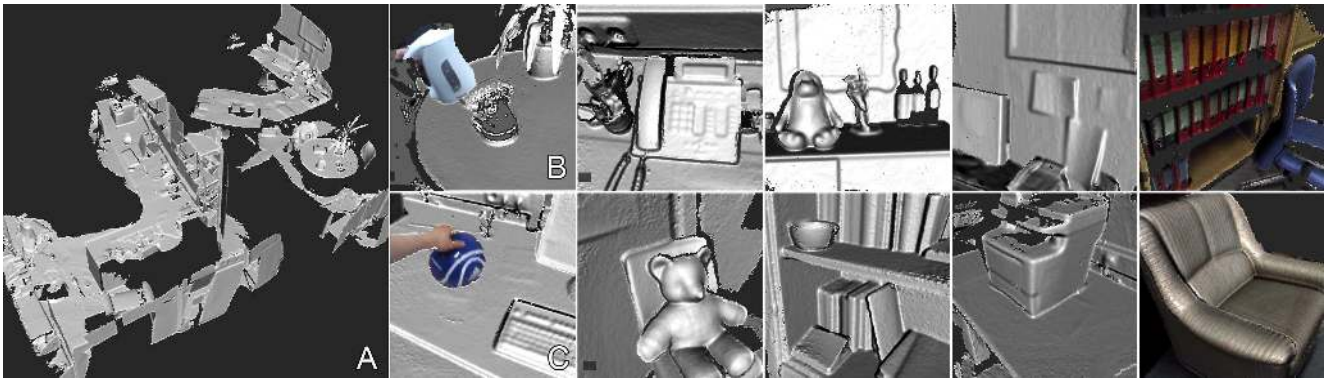

Figure 7. The *Large Office* scene, consisting of two large rooms and connecting corridors. A: overview; B and C: dynamically moving objects during acquisition; Note the millimeter scale of the phone's keypad. Other close-ups are also shown (right column: RGB textured).

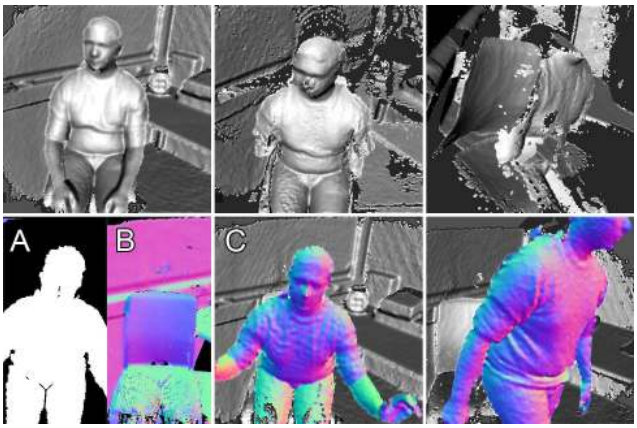

Figure 8. The *Moving Person* scene. Person sits on a chair, is reconstructed, and then moves. Dynamic parts occupy much of the field-of-view and cause ICP errors with previous approaches (top row). Segmenting the dynamics (A) and ignoring them during pose estimation (B) allows increased robustness (bottom row).

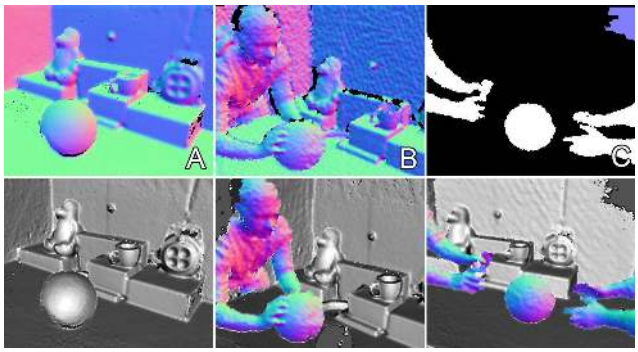

Figure 9. The *Ballgame* scene consists of two people moving a ball across a table. A: global model colored with surface normals; B: raw input data of the previously static ball being picked up; C: segmentation of dynamic parts; Bottom row: reconstructed result (model points + dynamic parts).

Figure 10. A: the *PMD* scene acquired with a PMD ToF camera. B and C: close-ups using per-pixel intensity values for coloring.

the point level: each person is recognized as dynamic even if only parts of their bodies are actually moving. Static objects that start moving are marked as dynamic and their model points are demoted to unstable status, while dynamic objects that stop moving eventually reach stable status in the global model when the observed points gain enough confidence.

Most scenes shown throughout this paper were recorded with a Microsoft Kinect camera in near mode, but our method is agnostic to the type of sensor used. Fig. 10 shows an example scene recorded with a PMD CamBoard ToF camera, which exhibits significantly different noise and error characteristics [8]. In this example, we used the per-pixel amplitude information provided by PMD sensors in the computation of the sample confidence α (see Sec. 4.2).

8. Conclusion

We have presented a new system for online 3D reconstruction which demonstrates new capabilities beyond the state-of-art. Our system has been explicitly designed to allow for a single point-based representation to be used throughout our pipeline, which closely fits the sensor input, and is amenable to rendering (for visualization and data association) through the standard graphics pipeline.

Despite the lack of a spatial data structure, our system still captures many benefits of volumetric fusion, allowing for accumulation of denoised 3D models over time that exploit redundant samples, model measurement uncertainty, and make no topological assumptions. This is achieved using a new *point-based fusion* method based on [2]. Reconstructions at this scale, quality, speed and with the ability to deal with scene motion and dynamic updates have yet to be demonstrated by other point-based methods, and are core contributions of our work.

There are many areas for future work. For example, whilst our system scales to large scenes, there is the additional possibility of adding mechanisms for streaming subset of points (from GPU to CPU) especially once they are significantly far away from the current pose. This would help increase performance and clearly the point-based data would be low overhead in terms of CPU-GPU bandwidth. Another issue is sensor drift, which we do not currently tackle, instead focusing on the data representation. Drift in larger environments can become an issue and remains an interesting direction for future work. Here again the point-based representation might be more amenable to correction after loop closure detection, rather than resampling a dense voxel grid.

Acknowledgements

This research has partly been funded by the German Research Foundation (DFG), grant GRK-1564 Imaging New Modalities, and by the FP7 EU collaborative project BEAMING (248620). We thank Jens Orthmann for his work on the GPU framework osgCompute.

References

[1] P. Besl and N. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. and Mach. Intell.*, 14(2):239–256, 1992. 1

[2] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. Comp. Graph. & Interact. Techn.*, pages 303–312, 1996. 1, 2, 4, 7

[3] D. Gallup, M. Pollefeys, and J.-M. Frahm. 3d reconstruction using an n-layer heightmap. In *Pattern Recognition*, pages 1–10. Springer, 2010. 2

[4] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *Int. J. Robotics Research*, 31:647–663, Apr. 2012. 2

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics (Proc. SIGGRAPH)*, 26(2), 1992. 1

[6] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proc. ACM Symp. User Interface Softw. & Tech.*, pages 559–568, 2011. 1, 2, 5, 6

[7] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. EG Symp. Geom. Proc.*, 2006. 1

[8] A. Kolb, E. Barth, R. Koch, and R. Larsen. Time-of-flight cameras in computer graphics. *Computer Graphics Forum*, 29(1):141–159, 2010. 7

[9] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, et al. The digital michelangelo project: 3D scanning of large statues. In *Proc. Comp. Graph & Interact. Techn.*, pages 131–144, 2000. 1

[10] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987. 1

[11] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. IEEE Int. Symp. Mixed and Augm. Reality*, pages 127–136, 2011. 1, 3, 4

[12] R. Newcombe, S. Lovegrove, and A. Davison. DTAM: Dense tracking and mapping in real-time. In *Proc. IEEE Int. Conf. Comp. Vision*, pages 2320–2327, 2011. 1

[13] C. Nguyen, S. Izadi, and D. Lovell. Modeling Kinect sensor noise for improved 3D reconstruction and tracking. In *Proc. Int. Conf. 3D Imaging, Modeling, Processing, Vis. & Transmission*, pages 524–530, 2012. 4, 6

[14] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *Proc. IEEE Vis.*, pages 233–238. IEEE, 1998. 1

[15] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proc. Conf. Comp. Graphics & Interact. Techn.*, pages 335–342, 2000. 2

[16] M. Pollefeys, D. Nistér, J. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. Kim, P. Merrell, et al. Detailed real-time urban 3D reconstruction from video. *Int. J. Comp. Vision*, 78(2):143–167, 2008. 1

[17] H. Roth and M. Vona. Moving volume KinectFusion. In *British Machine Vision Conf.*, 2012. 1

[18] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3D model acquisition. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 21(3):438–446, 2002. 2

[19] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. IEEE Conf. Comp. Vision & Pat. Rec.*, volume 1, pages 519–528. IEEE, 2006. 1

[20] J. Stückler and S. Behnke. Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D cameras. In *Proc. IEEE Int. Conf. Multisensor Fusion & Information Integration*, pages 162–167, 2012. 2

[21] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proc. Int. Conf. Computer Vision*, pages 839–846, 1998. 3

[22] T. Weise, T. Wismer, B. Leibe, and L. Van Gool. In-hand scanning with online loop closure. In *Proc. IEEE Int. Conf. Computer Vision Workshops*, pages 1630–1637, 2009. 2, 3

[23] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended KinectFusion. Technical report, CSAIL, MIT, 2012. 1

[24] C. Yang and G. Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992. 1

[25] C. Zach. Fast and high quality fusion of depth maps. In *Proc. Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT)*, volume 1, 2008. 1

[26] M. Zeng, F. Zhao, J. Zheng, and X. Liu. Octree-based fusion for realtime 3D reconstruction. *Graph. Models*, 75(3):126 – 136, 2013. 1

[27] M. Zwicker, H. Pfister., J. V. Baar, and M. Gross. Surface splatting. In *Computer Graphics (Proc. SIGGRAPH)*, pages 371–378, 2001. 4