Real-time RGB-D Mapping and 3-D Modeling on the GPU using the Random Ball Cover Data Structure

Dominik Neumann[1], Felix Lugauer[1], Sebastian Bauer[1], Jakob Wasza[1], Joachim Hornegger[1,2]

[1]Pattern Recognition Lab, Department of Computer Science
[2]Erlangen Graduate School in Advanced Optical Technologies (SAOT)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

sebastian.bauer@cs.fau.de

Abstract

The modeling of three-dimensional scene geometry from temporal point cloud streams is of particular interest for a variety of computer vision applications. With the advent of RGB-D imaging devices that deliver dense, metric and textured 6-D data in real-time, on-the-fly reconstruction of static environments has come into reach. In this paper, we propose a system for real-time point cloud mapping based on an efficient implementation of the iterative closest point (ICP) algorithm on the graphics processing unit (GPU). In order to achieve robust mappings at real-time performance, our nearest neighbor search evaluates both geometric and photometric information in a direct manner. For acceleration of the search space traversal, we exploit the inherent computing parallelism of GPUs. In this work, we have investigated the fitness of the random ball cover (RBC) data structure and search algorithm, originally proposed for high-dimensional problems, for 6-D data. In particular, we introduce a scheme that enables both fast RBC construction and queries. The proposed system is validated on an indoor scene modeling scenario. For dense data from the Microsoft Kinect sensor (640×480 px), our implementation achieved ICP runtimes of < 20 ms on an off-the-shelf consumer GPU.

1. Introduction

In the past, the acquisition of dense 3-D range data was both tedious, time consuming and expensive, hence, hindering a widespread application. Lately, advances in RGB-D sensor design have rendered metric 3-D surface acquisition at convenient resolutions (up to 300k points) and framerates (up to 40 Hz) possible, holding potential for a variety of applications where real-time demands form a key aspect. The advent of Microsoft's Kinect [11], with more than ten million sales within a few months, has caused a furor in the field of consumer electronics. With the introduction of affordable hardware, 3-D perception is gaining popularity across a wide range of domains, such as computer gaming and home entertainment, augmented reality [4], medical engineering [2], robotic navigation and collision avoidance [22, 23].

We address the field of 3-D environment and model reconstruction, with manifold practical fields of applications. Among others, 3-D modeling is a key component for the construction of environment maps in robot or vehicle navigation [13, 18], acquisition of virtual 3-D models from real objects, and digitalization of heritage objects for restoration planning or archival storage [8]. In particular, in the field of robotics, there is an increasing interest in both 3-D environment reconstruction and simultaneous localization and mapping (SLAM) solutions [1, 5, 19]. For instance, in the recent Robot Operating System (ROS) contest, an RGB-D-SLAM implementation for Microsoft Kinect ranked first in the category *most useful* [9].

However, only few existing approaches have achieved interactive framerates [9, 10, 13, 15]. Huhle et al. proposed a system for on-the-fly 3-D scene modeling using a low resolution Time-of-Flight camera (160×120 pixels), typically achieving per-frame runtimes of > 2 s [15]. Engelhard et al. presented similar runtimes on Kinect data for an ICP-based RGB-D SLAM framework [9]. The RGB-D mapping framework of Henry et al. performs ICP registration in an average of 500 ms [13]. Only recently, a workshop demo of ongoing work by Fioraio and Konolige has indicated real-time framerates for a geometric ICP variant [10].

In this paper, we propose a framework that is capable of mapping point cloud data streams on-the-fly, enabling real-time 3-D scene modeling. For this reason, we have implemented a hybrid 6-D ICP variant that performs the alignment by jointly optimizing over both photometric appear-

ance and geometric shape matching [16]. In order to allow for on-the-fly processing, the corpus of the framework is implemented on the GPU. For the nearest neighbor search, being the bottleneck in the majority of previous ICP implementations, we propose the use of a data structure that is specifically designed to benefit from the inherent computing parallelism of GPU data processing. In this work, we have investigated the fitness of the random ball cover (RBC) search algorithm [6, 7] for low-dimensional 6-D data. In particular, trading accuracy against runtime, we introduce a modified approximative RBC variant that is optimized in terms of performance.

The remainder of the paper is organized as follows. In Sec. 2, we review relevant literature. We present our method for photogeometric 3-D mapping in Sec. 3 and discuss the evaluation results in Sec. 5. Eventually, we draw a conclusion in Sec. 6.

2. Related Work

More than a decade ago, Johnson and Kang were the first that proposed the incorporation of photometric information into the ICP framework (*Color-ICP*) in order to improve its robustness [3, 16]. The basic idea is that photometric information can compensate for regions with non-salient topologies, whereas geometric information can guide the pose estimation for faintly textured regions. Recently, modifications have been proposed that try to accelerate the nearest neighbor search by pruning the search space w.r.t. photometrically dissimilar points [8, 17]. However, this reduction typically comes with a loss in robustness.

Since modern RGB-D devices produce and propagate an immense data stream (up to the scale of 500 MBit/s), efficient implementations are inevitable in order to fulfill real-time constraints. For the ICP algorithm in general, a comprehensive survey of efficient implementation variants was given by Rusinkiewicz and Levoy [21]. However, the survey does not include hardware acceleration techniques. For the nearest neighbor search, being a major bottleneck in terms of runtime, CPU architectures have shown to benefit from space-partitioning data structures like k-d trees. In contrast to algorithmic improvements, hardware acceleration techniques have recently drawn the attention of the community. Garcia et al. have shown that a GPU-based brute-force implementation outperforms a CPU-based k-d tree [12]. The reason for this effect lies in the fact that the brute-force primitive can be interpreted as a matrix-matrix multiplication. This operation can be parallelized very efficiently on the GPU.

GPU implementations of traditional acceleration structures are challenging due to the non-parallel and recursive traversal nature of the underlying data structures. Qiu et al. [20] achieved excellent framerates for GPU based k-d tree queries. However, the construction of the tree is performed on the CPU, thus decreasing performance. Only recently, space-partitioning strategies that are specifically designed for GPU architectures have been addressed. A very promising approach is the random ball cover proposed by Cayton [6, 7]. The basic principle behind the RBC is a two-tier nearest neighbor search utilizing the brute-force primitive.

3. Methods

The proposed framework is composed of three stages (see Fig. 1). In an initial stage, the sensor data (orthogonal distances + photometric color information) are transferred to the GPU, where the corpus of the pipeline is executed. First, the transformation from the 2-D sensor domain to 3-D world coordinates and data preprocessing is performed (Sec. 3.1). Second, based on a set of extracted landmarks, a color ICP variant (Sec. 3.2) is applied. Third and last, the current point cloud is attached to the model based on the estimated transformation. Our method exploits the arithmetic power of modern GPUs for efficient nearest neighbor search with an inherently parallel data structure and query framework (RBC, Sec. 3.3).

3.1. Data Preprocessing on the GPU

The Kinect device acquires RGB-D data of VGA resolution (640×480 px) at 30 Hz. With regard to real-time constraints and regardless of the specific application, this spatial and temporal data density poses a challenge to data processing solutions. Hence, in addition to the actual point cloud alignment, we have extended our framework to perform on-the-fly RGB-D data preprocessing in a highly par-

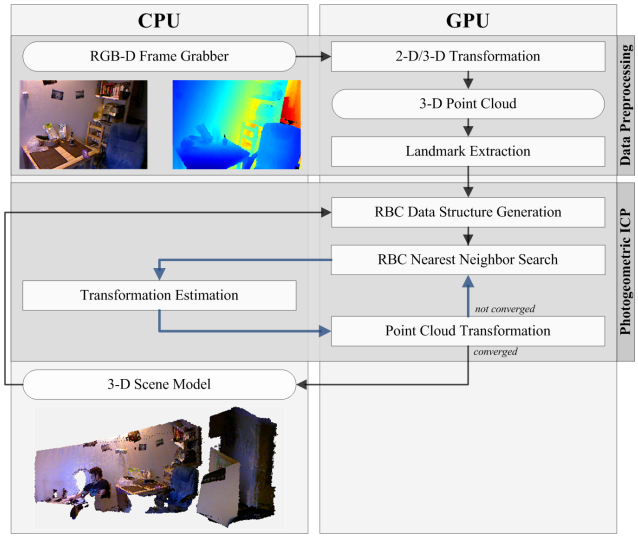


Figure 1. Flowchart of the 3-D scene reconstruction framework. Note that the corpus of the computation (including ICP) is outsourced to the GPU.

allel and efficient manner on the GPU [24]. First, the depth measurements delivered by Microsoft Kinect are to be transformed to the 3-D world coordinate system. Indeed, for each point $\boldsymbol{x}_c \in \mathbb{R}^2$ on the camera plane, its depth value $z(\boldsymbol{x}_c)$ describes a world coordinate position vector $\boldsymbol{x}_w \in \mathbb{R}^3$. In homogeneous coordinates, this transformation can be denoted as:

$$\begin{pmatrix} \boldsymbol{x}_{w,1} \\ \boldsymbol{x}_{w,2} \\ \boldsymbol{x}_{w,3} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{z(\boldsymbol{x}_c)}{f_x} & 0 & 0 \\ 0 & \frac{z(\boldsymbol{x}_c)}{f_y} & 0 \\ 0 & 0 & z(\boldsymbol{x}_c) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_{c,1} \\ \boldsymbol{x}_{c,2} \\ 1 \end{pmatrix} , \quad (1)$$

where f_x, f_y denote the focal length. This transformation may be computed independently for each pixel, thus fitting perfectly for parallel processing on the GPU (see Sec. 5.2).

Nomenclature Following Eq. 1, let us introduce the notation for the remainder of this section. Let $\tilde{\mathcal{M}}$ denote a (moving) set of template points $\tilde{\mathcal{M}} = \{\boldsymbol{m}\}$, where $\boldsymbol{m} \in \mathbb{R}^6$ concatenates a point's geometric and photometric information $\boldsymbol{m}_g \in \mathbb{R}^3$ and $\boldsymbol{m}_p \in \mathbb{R}^3$:

$$\boldsymbol{m} = \begin{pmatrix} \boldsymbol{m}_g \\ \boldsymbol{m}_p \end{pmatrix} . \quad (2)$$

Below, the indexes g and p denote that only the geometric or photometric part is considered. In order to compensate for inhomogeneities due to varying illumination conditions, the photometric information is transformed to the normalized RGB space, hence $\boldsymbol{m}_p = (r+b+g)^{-1}(r,g,b)^\top$. In analogy to $\tilde{\mathcal{M}}$, let $\tilde{\mathcal{F}} = \{\boldsymbol{f}\}$ denote a (fixed) set of $|\tilde{\mathcal{F}}|$ reference points $\boldsymbol{f} \in \mathbb{R}^6$, where $\boldsymbol{f}^\top = (\boldsymbol{f}_g^\top, \boldsymbol{f}_p^\top)$.

Considering the application for 3-D scene modeling using a hand-held and real-time RGB-D device, we assume that the pose of the acquisition device changes smoothly over successive frames. This implies that a portion of the scene that was captured in the previous frame $\tilde{\mathcal{F}}$ is no longer visible in the current data $\tilde{\mathcal{M}}$ and vice versa. Facing these issues, we heuristically clip the set of points that are located outside of the central sub-volume of the 3-D bounding box of $\tilde{\mathcal{M}}_g = \{\boldsymbol{m}_g\}$ in order to improve the robustness of ICP alignment. This clipping is performed in conjunction with the extraction of the sparse sets of ICP landmarks, denoted $\mathcal{M} \subset \tilde{\mathcal{M}}$ and $\mathcal{F} \subset \tilde{\mathcal{F}}$.

3.2. Photogeometric ICP Framework

The ICP algorithm is state-of-the-art for the rigid alignment of 3-D point clouds [3, 21]. It estimates the optimal rigid transformation $(\boldsymbol{R}, \boldsymbol{t})$ that brings $\mathcal{M}$ in congruence with $\mathcal{F}$, where $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ denotes a rotation matrix and $\boldsymbol{t} \in \mathbb{R}^3$ a translation vector. Below, we outline the essential steps of our *photogeometric* ICP variant incorporating both geometric and photometric information.

Based on a given initial estimation $(\boldsymbol{R}^0, \boldsymbol{t}^0)$, the ICP iteratively refines the transformation by minimizing the distance between $\mathcal{M}$ and $\mathcal{F}$ w.r.t. a metric d. In the geometric case, the distance d between an individual point $\boldsymbol{m}_g$ and the set of reference points $\mathcal{F}_g = \{\boldsymbol{f}_g\}$ is defined as:

$$d(\boldsymbol{m}_g, \mathcal{F}_g) = \min_{\boldsymbol{f}_g \in \mathcal{F}_g} \|\boldsymbol{f}_g - \boldsymbol{m}_g\|_2^2 , \quad (3)$$

where $\|\cdot\|_2$ denotes the Euclidean norm. In order to incorporate the additional photometric information available with modern RGB-D sensors, d can be modified to:

$$d(\boldsymbol{m}, \mathcal{F}) = \min_{\boldsymbol{f} \in \mathcal{F}} \left(\|\boldsymbol{f}_g - \boldsymbol{m}_g\|_2^2 + \alpha \|\boldsymbol{f}_p - \boldsymbol{m}_p\|_2^2 \right) , \quad (4)$$

where α is a non-negative constant weighting the influence of the photometric information. The benefit of this hybrid approach is that photometric information compensates for regions with non-salient surface topology, and topology information compensates for faintly textured regions or photometric inconsistencies due to varying illumination. The point in $\mathcal{F}$ that yields the minimum distance is denoted $\boldsymbol{y}$:

$$\boldsymbol{y} = \arg\min_{\boldsymbol{f} \in \mathcal{F}} \left(\|\boldsymbol{f}_g - \boldsymbol{m}_g\|_2^2 + \alpha \|\boldsymbol{f}_p - \boldsymbol{m}_p\|_2^2 \right) . \quad (5)$$

The evaluation of Eq. 5 $\forall\, \boldsymbol{m} \in \mathcal{M}$ eventually yields a set of nearest neighbors $Y = \{\boldsymbol{y}\}$.

For the k-th ICP iteration, based on the corresponding sets of points $(\mathcal{M}_g^k, Y_g^k)$, the transformation $(\hat{\boldsymbol{R}}^k, \hat{\boldsymbol{t}}^k)$ can be estimated in a least-squares sense using a unit quaternion optimizer [14]:

$$(\hat{\boldsymbol{R}}^k, \hat{\boldsymbol{t}}^k) = \arg\min_{\boldsymbol{R}^k, \boldsymbol{t}^k} \frac{1}{|\mathcal{M}_g^k|} \sum_{\mathcal{M}_g^k, Y_g^k} \|(\boldsymbol{R}^k \boldsymbol{m}_g^k + \boldsymbol{t}^k) - \boldsymbol{y}_g^k\|_2^2 .$$

After each iteration, the solution $(\boldsymbol{R}, \boldsymbol{t})$ is accumulated,

$$\boldsymbol{R} = \hat{\boldsymbol{R}}^k \boldsymbol{R} , \qquad \boldsymbol{t} = \hat{\boldsymbol{R}}^k \boldsymbol{t} + \hat{\boldsymbol{t}}^k , \quad (6)$$

and $\mathcal{M}_g^k$ is updated according to $\boldsymbol{m}_g^k = \boldsymbol{R}\boldsymbol{m}_g + \boldsymbol{t}$. The two stages of first finding the set of nearest neighbors Y^k and then estimating the optimal transformation for the correspondences $(\mathcal{M}_g^k, Y_g^k)$ are repeated iteratively until a convergence criterion is reached.

3.3. 6-D Nearest Neighbor Search using RBC

The random ball cover (RBC) [6, 7] is a novel data structure for efficient nearest neighbor (NN) search on the GPU. By design, it exploits the parallel architecture of modern graphics cards hardware. In particular, both the construction of the RBC and dataset queries are performed using brute-force (BF) primitives. Expressed as a matrix-matrix

multiplication, the BF search can be performed in a highly efficient manner on the GPU.

The RBC data structure relies on randomly selected points $\boldsymbol{r} \in \mathcal{F}$, called *representatives*. Each of them manages a local subset of $\mathcal{F}$ around $\boldsymbol{r}$. This indirection creates a hierarchy in the database such that a query is processed by (i) searching the nearest neighbor(s) among the set of representatives and (ii) performing another search for the subset of entries managed by $\boldsymbol{r}$. This two-tier approach outperforms a global BF search due to the fact that each of the two successive stages explore a heavily pruned search space.

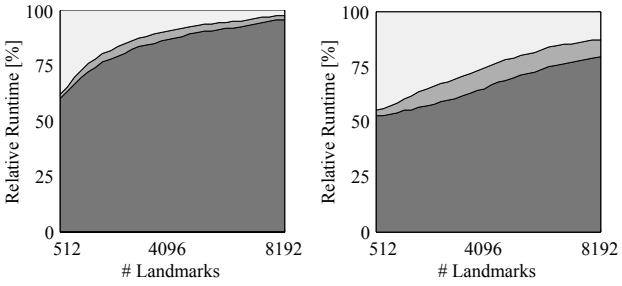
In this work, we have investigated the fitness of the RBC, originally proposed for high-dimensional spaces, for acceleration of the 6-D nearest neighbor search of our photogeometric ICP. Optimizing this particular ICP stage is motivated by the fact that it is a major bottleneck (see Sec. 5.2).

Cayton proposed two alternative RBC search strategies [7]. The *exact* search is an appropriate choice when the exact nearest neighbor is required. Else, if a small error may be tolerated, the probabilistic *one-shot* search is typically faster. Originally, in order to set up the *one-shot* data structure, the representatives are chosen at random, and each $\boldsymbol{r}$ contains the s closest database elements. Depending on s, points typically belong to more than one $\boldsymbol{r}$.

However, this implies a sorting of entries – hindering a high degree of parallelization for implementation on the GPU – or the need for multiple BF runs [6]. Hence, we introduce a modified version of the *one-shot* approach that is even further optimized in terms of performance. In particular, we have simplified the RBC construction down to a single BF search, trading accuracy against runtime: First, we extract a random set of representatives $\mathcal{R} = \{\boldsymbol{r}\}$ out of $\mathcal{F}$. Second, each representative $\boldsymbol{r}$ is assigned a local subset of $\mathcal{F}$. This is done in an inverse manner by simply computing the nearest representative $\boldsymbol{r}$ for each point $\boldsymbol{f} \in \mathcal{F}$. The query scheme of our modified *one-shot* RBC is consistent with the original approach and can be performed very efficiently using two subsequent BF runs [7]. Please note that our modified *one-shot* RBC is an approximative nearest neighbor search algorithm (see Sec. 5.2). The scheme can be applied to arbitrary dimensional data.

4. Implementation Details

Non-overlapping regions in subsequent frames may pose a challenge for the alignment process, and thus are obviated by clipping (see Sec. 3.1). In addition, an opposite effect may occur in the region of overlap. Let us assume that the camera maintains a static pose over time. In the presence of noise, repeatedly sampling the same portion of a scene will result in varying point clouds. This might lead to an error propagation when accumulating the estimated transformations over consecutive frames (Eq. 6). In order to overcome these effects, we measure the degree of overlap between consecutive frames by computing the L^1-distance of its depth histograms. Using this dissimilarity metric, the current RGB-D data will be discarded for mapping when the distance is below an empirically set threshold.



(a) BF (b) Our RBC

Figure 2. Relative runtime partitioning for a single ICP iteration including nearest neighbor search (dark gray), transformation estimation (medium gray) and transformation execution (light gray). Please note that the transformation estimation is performed on the CPU in the benchmarked implementation.

Regarding the quality of point cloud alignment, we observed a strong impact of outliers. Hence, we heuristically discard 10% of the correspondence pairs with the largest distance to their nearest neighbors. Please note that for an iterative scheme such as the ICP algorithm, this might influence the convergence behavior. However, we did not observe a significant effect in daily practice.

5. Experiments and Results

We have evaluated the proposed framework for on-the-fly 3-D modeling of real data (640×480 px, 30 Hz) from a hand-held Microsoft Kinect. Below, first we present qualitative results for an indoor scene mapping. Second, being a major focus of this system, we demonstrate its real-time capability in a comprehensive performance study. Third, we compare our approximative RBC variant to an exact nearest neighbor search. For all experiments, the number of representatives was set to $|\mathcal{R}| = \sqrt{|\mathcal{F}|}$ [7], if not stated otherwise. The performance study was conducted on an off-the-shelf consumer desktop computer running an NVIDIA GeForce GTX 460 GPU and an Intel Core 2 Quad Q9550 CPU. The GPU framework is implemented using CUDA[1].

5.1. Qualitative Results

Fig. 6 depicts qualitative results for an indoor scene modeling scenario. The sequence of point clouds was aligned on-the-fly. Please note that the proposed framework could also be used for a 3-D model digitalization scenario by moving the camera around an object. For this application, we

[1]The source code (C++/CUDA) of the proposed photogeometric ICP using our RBC variant is available from the authors for non-commercial research purposes.

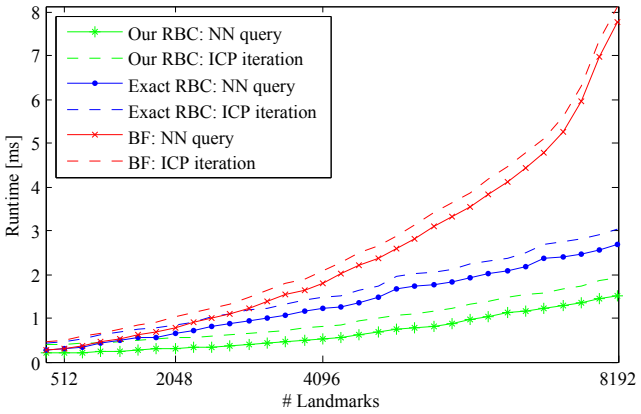


Figure 3. Comparison of the average runtime for a NN search based on a GPU BF primitive, the exact RBC and our optimized approximative RBC variant as described in Sec. 3.3. Note that our RBC approach outperforms the exact RBC up to a factor of two whereas the BF primitive scales quadratically.

typically modify our preprocessing pipeline in a way that points beyond a certain depth are ignored for the alignment procedure.

5.2. Performance Study

As stated before, the corpus of the proposed framework including both preprocessing and mapping (photogeometric ICP using our RBC variant) is executed on the GPU, recall Fig. 1. This section presents quantitative results for individual steps of the processing pipeline.

Data Preprocessing The computation of 3-D world coordinates from the measured depth values (see Sec. 3.1) takes < 1 ms for VGA resolution Kinect data, including CPU-GPU memory transfer of the RGB-D data. The subsequent clipping and landmark extraction for $\mathcal{M}$ and $\mathcal{F}$ depends on $|\mathcal{M}| = |\mathcal{F}|$, denoting the number of landmarks (LMs), with typical runtimes of < 0.3 ms. Hence, data preprocessing assumes a minor role.

# LMs	$\lvert\mathcal{R}\rvert$	t_{ICP}	$t_{\text{RBC,init}}$	# Iterations
512	23	6.8 ms	0.33 ms	15.4
1024	32	12.0 ms	0.41 ms	26.6
2048	**45**	**16.3 ms**	**0.56 ms**	**26.1**
4096	64	32.6 ms	0.91 ms	32.7
8192	91	84.7 ms	1.59 ms	38.8

Table 1. Runtimes for initialization of the RBC data structure ($t_{\text{RBC,init}}$) and ICP execution (t_{ICP}), for varying number of landmarks, $|\mathcal{R}| = \sqrt{|\mathcal{F}|}$. Given are average runtimes for modeling a typical indoor scene. In addition, the average number of ICP iterations until convergence is stated.

ICP using RBC Being the cornerstone of our framework, we have investigated the performance of our GPU-based ICP/RBC implementation in detail. A single ICP iteration consists of three steps: nearest neighbor search using RBC, transformation estimation and the transformation itself. With an increasing number of landmarks, the nearest neighbor search dominates the runtime considerably. This is illustrated in Fig. 2, where we opposed the runtime partitioning of a BF implementation on the GPU. Fig. 3 compares absolute runtimes for a single nearest neighbor query and ICP iteration, respectively. Our modified approximative RBC outperformed both a BF search and our reference implementation of Cayton's exact RBC. Practical runtimes of the method are given in Table 1, depicting values for a typical indoor scene mapping. As a performance indicator, let us refer to the runtime of 16.3 ms for 2048 landmarks, being our default configuration for Kinect data. In post-processing, the estimated transformation is applied to $\tilde{\mathcal{M}}_g$ (0.2 ms), which is then re-transfered to CPU memory (2.4 ms).

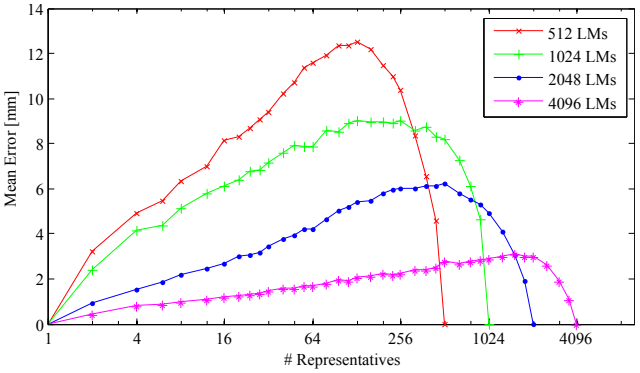


Figure 4. Evaluation of the influence of $|\mathcal{R}|$ on accuracy, for varying number of landmarks. Given is the mean Euclidean distance [mm] between the mapped points $\hat{\boldsymbol{m}}_{\text{RBC}}$ and $\hat{\boldsymbol{m}}_{\text{BF}}$. Note the semi-log scale.

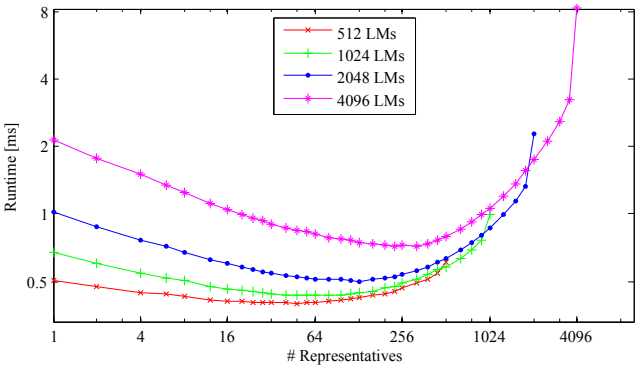


Figure 5. Runtimes of a single ICP iteration, for varying number of landmarks and representatives. Note the logarithmic scale.

Figure 6. On-the-fly 3-D reconstruction results for two indoor mapping scenarios, shown from three distinct viewpoints each. The datasets consist of 44 frames (first row) and 61 frames (second row) and both were acquired with a hand-held Microsoft Kinect camera.

Approximative RBC As motivated in Sec. 3.3, our approximative RBC nearest neighbor search sacrifices exactness for a runtime speedup. We quantitatively investigated the error that results from our approximate nearest neighbor search compared to an exact BF scheme, considering the aligned point clouds $\hat{\mathcal{M}}_{\text{RBC}}$ and $\hat{\mathcal{M}}_{\text{BF}}$, see Fig. 4. The error measures the mean pointwise Euclidean distance [mm] between the points $\hat{\boldsymbol{m}}_{\text{RBC}}$ and $\hat{\boldsymbol{m}}_{\text{BF}}$, being transformed w.r.t. different estimations for $(\boldsymbol{R}, \boldsymbol{t})$. Furthermore, we have related the runtime per ICP iteration to $|\mathcal{R}|$ (Fig. 5). Together, Fig. 4, 5 illustrate the trade-off between error and runtime, controlled by $|\mathcal{R}|$. Using our default configuration (2048 LMs) and Cayton's rule of thumb, $|\mathcal{R}| = \sqrt{|\mathcal{F}|}$, the mapping error is less than 5 mm. This is an acceptable scale for the applications considered here.

6. Discussion and Conclusions

In this paper, we have proposed a GPU framework for real-time mapping of textured point cloud streams enabling on-the-fly 3-D modeling with modern RGB-D imaging devices. Our quantitative RBC experiments demonstrate that using a data structure specifically designed to exploit the parallel computing power of GPUs is beneficial even for low-dimensional (6-D) data. Using our optimized approximative RBC for the photogeometric nearest neighbor search, our system achieves ICP runtimes of < 20 ms on an off-the-shelf consumer GPU, for Microsoft Kinect data (640×480 px). An extension of the proposed system for SLAM will be subject of our upcoming research.

References

[1] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part II state of the art. *Computational Complexity*, 13(3):1–10, 2006.

[2] S. Bauer, B. Berkels, J. Hornegger, and M. Rumpf. Joint ToF image denoising and registration with a CT surface in radiation therapy. In *Proceedings of International Conference on Scale Space and Variational Methods in Computer Vision*, volume 6667 of *LNCS*, pages 98–109. Springer, May 2011.

[3] P. Besl and N. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[4] T. Blum and N. Navab. Augmented Reality Magic Mirror using the Kinect, http://campar.in.tum.de, 2011.

[5] V. Castaneda, D. Mateus, and N. Navab. SLAM combining ToF and high-resolution cameras. In *Proceedings of IEEE Workshop on Applications of Computer Vision*, pages 672–678, Jan 2011.

[6] L. Cayton. A nearest neighbor data structure for graphics hardware. In *Proceedings of International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, 2010.

[7] L. Cayton. Accelerating nearest neighbor search on manycore systems. *Computing Research Repository*, abs/1103.2635, Mar 2011.

[8] S. Druon, M. Aldon, and A. Crosnier. Color constrained ICP for registration of large unstructured 3D color data sets. In *Proceedings of IEEE International Conference on Information Acquisition*, pages 249–255, Aug 2006.

[9] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3D visual SLAM with a hand-held RGB-D camera. In *Proceedings of RGB-D Workshop on 3D Perception in Robotics, European Robotics Forum*, 2011.

[10] N. Fioraio and K. Konolige. Realtime visual and point cloud SLAM. In *Proceedings of RGB-D Workshop: Advanced Reasoning with Depth Cameras, Robotics Science and Systems Conference*, 2011.

[11] J. Garcia and Z. Zalevsky. Range mapping using speckle decorrelation. US patent No. 7433024, 2008.

[12] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using GPU. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshop on Computer Vision on GPU*, pages 1–6, Jun 2008.

[13] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Proceedings of International Symposium on Experimental Robotics*, 2010.

[14] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, Apr 1987.

[15] B. Huhle, P. Jenke, and W. Strasser. On-the-fly scene acquisition with a handy multi-sensor system. *International Journal of Intelligent Systems Technologies and Applications*, 5:255–263, Nov 2008.

[16] A. Johnson and S. B. Kang. Registration and integration of textured 3-D data. In *Proceedings of International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pages 234–241, May 1997.

[17] J. H. Joung, K. H. An, J. W. Kang, M. J. Chung, and W. Yu. 3D environment reconstruction using modified color ICP algorithm by fusion of a camera and a 3D laser range finder. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3082–3088, Oct 2009.

[18] S. May, D. Droeschel, D. Holz, S. Fuchs, E. Malis, A. Nüchter, and J. Hertzberg. Three-dimensional mapping with time-of-flight cameras. *Journal of Field Robotics*, 26:934–965, Nov 2009.

[19] A. Nüchter, H. Surmann, K. Lingemann, J. Hertzberg, and S. Thrun. 6D SLAM with an application in autonomous mine mapping. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1998–2003, Apr 2004.

[20] D. Qiu, S. May, and A. Nüchter. GPU-accelerated nearest neighbor search for 3D registration. In *Proceedings of International Conference on Computer Vision Systems*, pages 194–203. Springer, Oct 2009.

[21] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings of International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[22] Starmac. The Stanford/Berkeley Testbed of Autonomous Rotorcraft for Multi-Agent Control, 2011. http://hybrid.eecs.berkeley.edu/starmac/.

[23] TUM. Kinect enabled robot workspace surveillance, 2011. http://www6.in.tum.de/Main/ResearchJahir.

[24] J. Wasza, S. Bauer, S. Haase, M. Schmid, S. Reichert, and J. Hornegger. RITK: The range imaging toolkit - a framework for 3-D range image stream processing. In *Proceedings of International Workshop on Vision, Modeling, and Visualization*, Oct 2011.