KinFu MOT: KinectFusion with Moving Objects Tracking

Michael Korn and Josef Pauli
Intelligent Systems Group, University of Duisburg-Essen, 47057 Duisburg, Germany

Keywords: 3D, GPU, Tracking, SLAM, Iterative Closest Point (ICP), Point Cloud Registration, Depth Cameras.

Abstract: Using a depth camera, the KinectFusion algorithm permits tracking the camera poses and building a dense 3D reconstruction of the environment simultaneously in real-time. We present an extension to this algorithm that allows additionally the concurrent tracking and reconstruction of several moving objects within the perceived environment. This is achieved through an expansion of the GPU processing pipeline by several new functionalities. Our system detects moving objects from the registration results and it creates a separate storing volume for such objects. Each object and the background are tracked and reconstructed individually. Since the size of an object is uncertain at the moment of detection the storing volume grows dynamically. Moreover, a sliding reduction method stabilizes the tracking of objects with ambiguous registrations. We provide experimental results showing the effects of our modified matching strategy. Furthermore, we demonstrate the system's ability to deal with three different challenging situations containing a moving robot.

1 INTRODUCTION

In recent years visual SLAM and 3D reconstruction methods with handheld sensors have reached a significant level of maturity. The requirements of real-time and robustness seem to be accomplished so comprehensively that they are not the weak point for several applications. Instead, further needs get more attention like the spatial extension of the perceivable volume with loop closure functionality (Endres et al., 2012) and dense mapping (Klein and Murray, 2007). With the emergence of highly available RGB-D cameras, in particular the Microsoft Kinect, the meaning of a dense map has developed further – together with the new potentials. In (Izadi et al., 2011; Newcombe et al., 2011) the KinectFusion algorithm (KinFu) was introduced and it has become a state-of-the-art method in the terms of robustness, real-time and density. Since KinFu accumulates all information obtained from the depth frames in a small ($3^3 m^3$) stationary voxel grid, the spatial scope is too small for several applications. These spatial limitations are already being tackled, too. With (Heredia and Favie, 2012) an open source KinFu derivative with a moving voxel grid is available. A similar approach is described in (Whelan et al., 2012). This paper also mentions ongoing work on SLAM pose-graph optimization and hence loop closing.

A large and least treated issue represents the presence of moving objects. On the one hand such objects can interfere with the camera movement reconstruction and force the algorithm to fail. On the other hand a moving object can lead to a gain of information. Such an object can be clearly segmented and reconstructed from different perspectives, independent of the camera movement. Furthermore, the object movement can be tracked which yields advantages for example in the field of autonomous robot navigation or articulated objects. This paper presents a way to make these benefits available by detecting the movement of rigid objects and creating a dense reconstruction of each object in a separate voxel grid.

Section 2 summarizes some required background knowledge concerning KinFu, section 3 describes our modifications to the matching step of KinFu and our pipeline extensions, section 4 details our experiments and results. In the final section we discuss our conclusions and future directions.

2 BACKGROUND

Our implementation builds on the open source publication of KinFu released by the Point-Cloud-Library (PCL) (Rusu and Cousins, 2011) which is based strictly on the original descriptions from (Izadi et al., 2011; Newcombe et al., 2011). In the next two subsections we point at the limitations of KinFu.

Korn M. and Pauli J..
KinFu MOT: KinectFusion with Moving Objects Tracking.
DOI: 10.5220/0005362106480657
In *Proceedings of the 10th International Conference on Computer Vision Theory and Applications* (VISAPP-2015), pages 648-657
ISBN: 978-989-758-091-8

2.1 KinectFusion

The KinFu algorithm generates a 3D reconstruction of the environment on a GPU in real-time by integrating all available depth maps from a depth sensor into a discretized Truncated Signed Distance Functions (TSDF) representation (Curless and Levoy, 1996). The measurements are collected in a voxel grid in which each voxel stores a truncated distance to the closest surface including a related weight that is proportional to the certainty of the stored value. To integrate the depth data into the voxel grid every incoming depth map is transformed into a vertex map and normal map pyramid. Another deduced vertex map and normal map pyramid is obtained by ray casting the voxel grid based on the last known camera pose. According to the camera view, the grid is sampled by rays searching in steps for zero crossings of the TSDF values. Both pyramids are registered by a point-to-plane ICP procedure and the resulting transformation determines the current camera pose. Subsequently, the voxel grid data is updated by iterating over all voxels and the projection of each voxel into the image plane of the camera. The new TSDF values are calculated using a weighted running average. Usually the weights are also truncated allowing the reconstruction of scenes with some degree of dynamic. Finally, the maps created by the raycaster are used to generate a rendered image of the implicit environment model.

Furthermore, the detection of outliers is in the scope of (Izadi et al., 2011) which is not a part of the PCL. The possibility is described to first reconstruct the background and then to use the ICP outliers to detect movement. They primarily intended to allow interaction with the background model, but the 3D reconstruction of the foreground object was also suggested. However, the prospects are restricted since the object cannot be reconstructed simultaneously with the background.

2.2 KinectFusion Limitations

KinFu is primarily limited to the reconstruction of a small static environment which fits into the $512 \times 512 \times 512$ voxel grid. Moving objects in the space of the perceived environment are not totally off-limits, but they cause outliers in the registration process and are able to interfere with the estimated camera poses. In the best case, as long as such objects are moved quickly, they will not become part of the environment model and they stay disregarded entirely. The other issue of a small size of the voxel grid is caused by the claim of reaching real-time capability and the fact

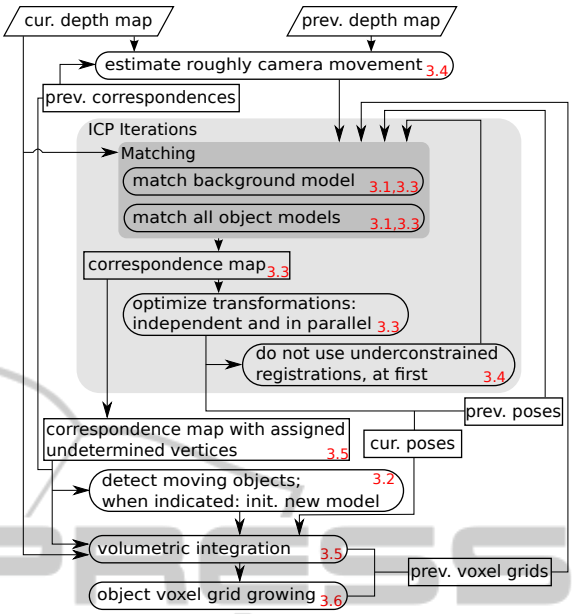

Figure 1: Overview of the processing pipeline which is introduced in section 3. Related subsection are indicated in red. Inputs as parallelograms, data as rounded rectangles and processing steps as rectangles.

that the runtimes of the raycast process and the depth map integrations process rely strongly on the voxel grid size. It is desirable to reduce the impact of the voxel grid size on the runtimes to increase the number of voxel. This would allow to cover a larger section of the environment or to enhance the resolution of the voxel grid. Furthermore, the targeted additional reconstruction of several moving objects leads to the need of being able to use a larger number of voxels to store the extra information.

3 CONTRIBUTION

In this section we first present an alternative matching method for the ICP. It is designed to deal with a larger number of voxels, since every object gets its own voxel grid in addition to the voxel grid of the background. This method is also able to deal with more dynamics in the scene due to the fact, that we do not sample the voxel grid based on old pose information. This is especially advantageous when the object and the camera move in opposed directions. In the remaining subsections, we introduce several extensions to the original KinFu GPU processing pipeline which permits to reconstruct the background and several moving rigid objects in parallel and in real-time. A simplified overview of the resulting pipeline is given in Figure 1.

3.1 Vertex Map to Model Matching

KinFu creates deduced vertex and normal maps from the voxel grid data and register these maps with the maps obtained from the Kinect. The sampling of the voxel grid causes unwanted information reduction. It may happen, that the depth measurements contain no adequate correspondence in the deduced data, due to missing points caused by occlusion, a slight offset of the measured area, imperfect data in the voxel grid or inaccurate sampling. Furthermore, ray casting the whole voxel grid is not always necessary and the effort can be reduced. A sample application is robot navigation in which no rendered image of the environment is needed. Even when a rendered image is desired, the viewpoint of the virtual camera can differ from the viewpoint of the kinect and consequently two raycast processes are required.

We introduce a method to avoid the registration with maps which are deduced from the voxelgrid. Instead, the maps obtained from the Kinect are matched directly against the model in the voxel grid. Our modification is based on the idea that the Kinect vertex map indicates possible correspondences between the latest data and the model. We conform as far as possible to the notation from (Izadi et al., 2011) in Algorithm 1 in order to make comparisons simple. Substantially, Algorithm 1 is a combination of a locally limited raycaster and a projective ICP algorithm. In parallel, each GPU thread walks in steps within a section of a single ray through the voxel grid. The threads search for an implicit surface within the voxel grid matching the related point from the vertex map obtained from the latest kinect frame. On the one hand, the step size $step = trunc\_dist \cdot 0.6$ must be small enough so that no zero crossing is missed. On the other hand, too small of steps or even a cell by cell handling downgrade the results significantly because of the noisy data. Obviously, larger steps have a smoothing effect and reduce the runtime, too. The rays are determined by the latest vertex map V and the best known Kinect pose T. After a new Kinect frame arrives, T is initialized with the physical global camera pose obtained from the previous frame and refined iteratively. Depending on the current Pose T and the vertex map V the rays run from the estimated camera position to the measured points. The search for correspondences must only take place along the rays in the neighborhood of the measured points restricted by the distance threshold for the ICP plus a tolerance depending on the step size. The exact surface intersection point is computed using a linear interpolation based on the trilinearly extracted TSDF values of the points g and g_{prev}. As might be expected from

Algorithm 1: Vertex Map to Model Matching.

Input: vertex map V, normal map N, voxel grid $grid$, latest transformation (camera pose) $T = [R|t]$ with rotation R and translation t, distance threshold for ICP $distThres$, step wide $step$

Output: point pairs incl. normal for each matched pixel u

1: **for** each valid pixel $u \in N$ **in parallel do**
2: $v_u^g \leftarrow T \cdot V(u)$
3: $ray^{dir} \leftarrow \text{normalize}(v_u^g - t)$
4: $ray^{start} \leftarrow v_u^g - ray^{dir} \cdot (distThres + step)$
5: $ray^{endLen} \leftarrow 2 \cdot (distThres + step)$
6: $ray^{len} \leftarrow \max(0, \text{lenToEnterVolume}(ray^{start}, ray^{dir}))$
7: **if** $ray^{len} >= ray^{endLen} - step$ **then**
8: exit
9: $g \leftarrow ray^{start} + ray^{dir} \cdot ray^{len}$
10: **while** $ray^{len} < ray^{endLen}$ **do**
11: $g_{prev} \leftarrow g$
12: $g \leftarrow ray^{start} + ray^{dir} \cdot (ray^{len} + step)$
13: **if** g left volume bounds **then**
14: exit
15: **if** zero crossing from $grid(g)$ to $grid(g_{prev})$ **then**
16: $l \leftarrow$ extract trilinear interpolated ray length
17: $l \leftarrow \min(\max(l, ray^{len}), ray^{len} + step)$
18: **if** $l < step$ **or** $l > (2 \cdot distThres + step)$ **then**
19: exit
20: $v_u^{model} \leftarrow ray^{start} + ray^{dir} \cdot l$
21: $n_u^g \leftarrow R \cdot N(u)$
22: **return** correspondence $\langle v_u^{model}, v_u^g, n_u^g \rangle$
23: $ray^{len} \leftarrow ray^{len} + step$

line 17 the trilinear interpolation of the TSDF data is susceptible to faults and the computed points are located sometimes several meters away from the correct position. We have followed the original PCL implementation regarding the interpolation, but since the interpolation is time-consuming and the benefit is questionable, this approach should be subject to further evaluation. After a final check, whether the distance threshold is not exceeded, the found correspondences consisting of vertex v_u^g, the related normal n^g and the computed intersection point between the ray and implicit surface v_u^{model} take part in the optimization process. The approaches to build the linear equation system and to solve it are based on the original KinFu paper and remain unchanged in this subsection.

The following part points at some conceptual differences between our ICP process and the original procedure. First of all we abstain from using a threshold between the normals extracted from the voxel grid and the normals calculated from the depth map. In our experiments we were not able to find a general benefit in such a threshold. On the contrary, we observed that in the PCL implementation too many correspondences are rejected because the normals differ more than the threshold of 20°. In some experiments

with our matching approach and a normal threshold, about 15% of mainly useful correspondences were rejected. Aside from the problems with noise and interpolation, mentioned above, the Kinect depth maps are disturbed by a vertical banding. In these areas the normal threshold rejects most correspondences even on homogeneous surfaces. The normal threshold would have to be softened so much that it should be removed right away. Furthermore, we use the normals calculated from the depth map and not the normals extracted from the voxel grid for the point-to-plane error metric. These modifications show small improvements on some datasets and small degradations on other datasets. Overall the results remain the same. However, the compution of the model normals is relatively expensive and can be avoided. Finally, we always use the full resolution of the kinect and we removed the multi-scale ICP alignment proposed in (Newcombe et al., 2011) since we could not see any advantage that justifies that additional effort in our experiments.

3.2 Object Initialization

A moving object can be recognized by outliers during the ICP registration. Unfortunately, in addition to noise, the projective ICP algorithm leads to a lot of outliers which must not be mistaken as movement. In particular, this ICP needs a large distance threshold, the default is 10cm. On the one hand, detecting movement only after a large displacement is not satisfactory and it may work only for fast objects since small displacements are gradually integrated into the background voxel grid. On the other hand, reducing the distance threshold has a significant adverse effect on the results. To consider both, we use a threshold of 10cm but in the last iteration we decrease it do 3.5cm. We estimate further potential outliers according to the point-to-plane metric $(v_u^{model} - v_u^g) \cdot n_u^g > 2\text{cm}$. These correspondences are used in the registration process, but also marked as outliers for further processing.

Subsequently, the neighborhood with a size of 51×51 pixels of each marked outlier o_u is investigated. If 90% of neighbor pixels are marked as outliers, o_u is marked as a moving object. In the next step, each outlier in the 51×51 neighborhood of a pixel marked as moving object, is marked as moving object, too.

If such pixels were detected, an initialization of a new moving object is taken into account. An initialization is only carried out if (a) a moving object was suspected in at least 5 of the sequential previous frames and (b) it has been more than 30 frames (1 second) since the last object initialization. Additionally, (c) the initialization is postponed by 30 extra frames after each object growing (see subsection 3.6). This indicates that an object is not recognized as a whole. In the case of an initialization, the bounding box for the marked vertices is determined to obtain the position and size of a new voxel grid. This voxel grid is enlarged by 10% and 10cm in each of the six directions to allow growing of the object model. Finally, the volumetric integration of the marked pixels is performed.

3.3 Matching and Pose Estimation

The pose estimation is distinguished from the original by the vertex map to model matching (Algorithm 1) of every voxel grid $voxel\_grid_i$ with $i \in \{bg\} \cup objectIDs$. Within seven iterations, first the correspondences between vertex map V and background voxel grid $voxel\_grid_{bg}$ are found and subsequently the correspondences between V and all object voxel grids. Since every vertex could be assigned to several voxel grids the best matching for each $V(u)$ is searched. For this purpose Algorithm 1 is extended by a correspondence map C within which the best assignation $C(u)$ for each $V(u)$ is stored. In the case of several matchings, the correspondence with the shortest point-to-plane distance $(v_u^{model} - v_u^g) \cdot n_u^g$ is chosen. As this value is anyway computed for the optimization process the additional effort for this approach is negligible.

The background model and each detected object have their own registration. Therefore, separate normal equation systems are created for the background and each detected object which results in different transformation estimations x_i:

$$(A_i^T A_i)x_i = A_i^T b_i \text{ , with } i \in \{bg\} \cup objectIDs \quad (1)$$

In contrast to the original approach, it is not possible to create normal equations for each block of the vertex map (The GPU implementation works in blocks with size 32×8 pixels.) in the first step and to sum this equations in a subsequent step. Before creating the normal equations, V must be matched with every model and the correspondences have to be fixed. Accordingly, we cannot store the equation systems of small blocks in a buffer, but each line $[A_i|b_i]$ is buffered. Since the assignation $C(u)$ of any voxel position u is distinct, only one common buffer $[A|b]$ for all models is sufficient. After this pure matching and preparation step, Equation 1 is settled up for $voxel\_grid_{bg}$ and each other $voxel\_grid_i$. $[A_i^T A_i | A_i^T b_i]_{6\times 7}$ consists of 27 independent elements because of symmetry. The performance of this approach is similar to the original since any of these elements of any object can be computed separately by

their own GPU block in parallel with tree-based reduction.

3.4 Sliding Reduction

A substantial drawback of the point-to-plane metric is the tendency to diverge, if the transformation between the model and the data is not fully constrained. A registration of a planar surface for example has three unconstrained degrees of freedom: translation in two directions along the plane and rotation around the plane normal. Neither of these change the point-to-plane error significantly and the resulting transformation may slide in each ICP iteration due to noise. This issue can already be observed in the original KinFu algorithm but since tracked objects are smaller and with less details than the background, especially right after the initialization, its importance increases. Additionally, the transformation between the current V and the background results from the camera movement. Whereas the transformation between V and an object results from the camera movement and the object movement. Due to the assumption of marginal movement the ICP starts the processing of a new frame with the last known transformations. Since the camera movement remains unconsidered during the first iteration, it flows into the initialization of the object transformation. This influence may have an impact on the final estimated transformation in such a way that a part of the camera movement is ascribed to the object movement. We deal with the sliding in two steps: first we compute a rough estimation of the camera movement for the initialization of the transformations. Furthermore, we use the updated transformation estimation from the ICP only after the last two iterations if it is a subject to an underconstrained registration.

Before the actual ICP we determine the transformation between the current V and the previous vertex map $\hat{V}$ based on the previous assignment of correspondences $\hat{C}$. There are no multiple iterations and as shown in Algorithm 2 this approach is simple compared to the original matching step. Since the first supposed transformation is the identity matrix we can avoid any coordinate transformation and also the perspective projection into the image coordinate system. In the further process we use the original method for creating the normal equations and computing the first estimation of the incremental transformation. It is used to correct the initial guess of the camera movement. Since the first optimization step has by far the largest impact, this improves the first object matching greatly.

Algorithm 2: Vertex Map V to $\hat{V}$ Matching.

1: **for** each pixel $u \in V$ **in parallel do**
2: **if** $\hat{C}(u) = backgroundID$ **and**
$V(u)$ valid **and**
$\hat{N}(u)$ valid **and**
$\|\hat{V}(u) - V(u)\| \leq distThres$ **then**
3: **return** correspondence $\langle \hat{V}(u), V(u), \hat{N}(u) \rangle$

Even with a perfect initialization, sliding cannot be prevented. (Gelfand et al., 2003) describes amongst others how the stability of a registration can be estimated reliably. A condition number $\frac{\lambda_1}{\lambda_6}$ can be computed from the eigenvalues $\lambda_1 \geq \ldots \geq \lambda_6$ of $A_i^T A_i$. A small condition number indicates at least one unconstrained degree of freedom and an unstable registration due to possible sliding. In our experiments with stable situations the condition number is around 100 and going far beyond 1000 when we observe sliding. Given this margin a reliable threshold can be easily found and we choose 1000. If this threshold is exceeded, a situation occurs where the result of the ICP is questionable and should not be used. Obviously, using wrong data should be avoided, however the object is probably still in movement and without registration the tracking will be lost after several frames. As a compromise, just the last two ICP iterations decide the object movement. Since the camera movement is quite well estimated from this point and the iterations are few, the sliding is within limits. Moreover, just two iterations are absolutely sufficient because the data do not contain enough information for a registration which needs to be approximated over many iterations. During the first five iterations the object is still part of the matching process to reduce false matchings of other objects or the background. In addition, the first six iterations are always started with respect to the most recent camera movement estimation.

3.5 Volumetric Integration

The information from each new frame is integrated into every voxel grid $voxel\_grid_i$ with $i \in \{bg\} \cup objectIDs$. This can be initiated in parallel since the voxel grids are independent. After the pose estimation step, the camera poses related to the coordinate systems of each $voxel\_grid_i$ are known. Furthermore, every pixel of the recent depth map is assigned to one particular $voxel\_grid_i$ based on the correspondences from C. In addition, a distance map D is prepared where $D(u)$ is the measured distance between the camera and the perceived surface at pixel u based on the raw depth map. Our integration method uses major parts from the PCL and the simplified concept is summarized in Algorithm 3. The changes compared with (Izadi et al., 2011) are located between the lines

Algorithm 3: Volumetric Integration.

for each $g = (x, y, 0) \in voxel\_grid_i$ **in parallel do**
  for each z with $g = (g_x, g_y, z) \in voxel\_grid_i$ **do**
    $v^g \leftarrow$ convert g from grid to global 3D position
    $v \leftarrow T_i^{-1} \cdot v^g$
    $u \leftarrow$ project v into image plane
    if $v_z > 0$ **and** $u \in D$ **and** $D(u)$ valid **then**
      $sdf \leftarrow D(u) - \|t_i - v^g\|$
      if $sdf \geq -trunc\_dist$ **then**
        if $C(u) = i$ **or**
        ($C(u)$ invalid **and** $C(u) = backgroundID$) **then**
          $tsdf \leftarrow \min(1, sdf/trunc\_dist)$
        else
          if $sdf < 0$ **then**
            continue
          $tsdf \leftarrow 1$
        read average tsdf and weight in $voxel\_grid_i$ at g
        compute and store new average tsdf and weight

8 and 14. The whole TSDF proposed in (Izadi et al., 2011) is only used in the case that the particular pixel is assigned by C to the current voxel grid or the correspondence of this pixel is unknown and the current voxel grid contains the background model. Elsewise the $tsdf$ value for the current integration is set to 1 for all voxel between the camera and the measured surface and the stored values behind the surface remain unchanged.

Although this procedure works in general and it is able to complete gradually a moving object which was initialized partially at the beginning, the rate of convergence is unsatisfactory. Usually, shortly after an initialization a cluster of pixels can be matched with the object. This cluster is surrounded with pixels which belong to the object but were matched with the background model since this information was integrated even before the movement was detected. This is a result of the projective ICP that matches a vertex with the nearest surface along the particular ray and not just with the nearest surface of the surrounding area. While the object area grows due to imperfect alignments of the depth maps and the voxel grid, the correspondences with the background model decline due to exceedances of the distance threshold. We use these unmatched pixels to accelerate the growing of the object.

Subsequent to the ICP process the correspondence map C is modified based on the vertex map V with Algorithm 4. We apply this method three times to reach more pixels and we use in our implementation a buffered C to ensure determinism of our whole approach.We investigate the neighborhood of any outlier and any potential outlier according to a large point-to-plane distance (see subsection 3.2). We search the nearest of the 8-connected neighbors with determined correspondence. Finally, we adopt the assignment if the distance between the original vertex and his neighbor is lower than a threshold. We use $distance\_scale = \frac{3}{focal\_length}$, where $focal\_length$ is the focal length expressed in pixel units. This scale multiplied with v_z yields to the triple distance of intersections of neighboring rays and the plane parallel to the x and y axis at $z = v_z$.

Algorithm 4: Assign Undetermined Vertices.

for each pixel $u \in V \setminus$ 1-pixel-border **in parallel do**
  if $V(u)$ valid **and** $C(u)$ invalid **or**
    u is potential outlier **then**
    $v \leftarrow V(u)$
    $buf^{dist} \leftarrow \infty$
    for each $\upsilon \in$ 8-connected neighbors of u **do**
      if $C(\upsilon)$ valid **then**
        $dist \leftarrow \|v - V(\upsilon)\|$
        if $dist < buf^{dist}$ **then**
          $buf^{dist} \leftarrow dist$
          $buf^{id} \leftarrow C(\upsilon)$
    if $dist < v_z \cdot distance\_scale$ **then**
      $C(u) \leftarrow buf^{id}$

3.6 Object Voxel Grid Growing

Due to an imperfect initialization of a moving object and the ability to grow during the volumetric integration, the volume capacity of an object voxel grid is often reached. The capacity of a voxel grid is reached, if at least one voxel of any of the six sides of the voxel grid was updated after the initialization and it contains a TSDF value < 1. This means the distance between the object model and the voxel grid border is $< trunc\_dist$. Right after every volumetric integration of an object the borders of the voxel grid were searched for such voxels. Each grid side containing such voxels is expended by 10% of the grid size in the related direction. For this purpose a further voxel grid with the new size is created. The content of the old voxel grid is copied into the new one and the extended areas are initialized. Subsequently, the old grid is replaced by the new grid and the old grid is released. Since an object voxel grid is usually small relative to the background model and all this is done on the GPU and just the device memory is involved, the time effort is not particularly great.

3.7 GPU Implementation

The PCL provides a KinFu version for academic research. There is just one CPU thread and the host code waits regularly for kernel calls. Because of this, runtimes can be measured well but the whole algorithm is much slower. With our modifications it is impossible to reach acceptable framerates under these

conditions. However, the main problem is not the runtime of our KinFu algorithm, but the duration for painting five windows (rendered background, one rendered object, depth image, correspondences illustration and an interactive examination window) instead of just three. We use three threads. One preparation thread for grabbing the depth images and creating the vertex and normal maps. One Thread for controlling and painting the visualization. And one execution thread for the actual GPU processing pipeline. Furthermore, the building of the normal equation systems and their download to host memory is parallelized with cuda streams. In the second phase the detection of outliers and moving objects and volumetric integrations including growing of all objects is parallelized. Our concept has two bottlenecks: First, the matching – due to finding distinct correspondences without the risk of race conditions. Secondly, the waiting for the ICP results since many following parts relay on them. This degree of parallelism is sufficient for now but to keep the code clear the potential had not yet been fully exhausted.

4 EVALUATION AND RESULTS

We measured the performance and SLAM accuracy of our matching strategy, comparing it with the original KinFu implementation based on three datasets with known ground truth.

Furthermore, we provide qualitative evaluations of the capabilities of our whole processing pipeline. For this purpose we chose three scenes recorded with a handheld Microsoft Kinect containing a moving robot in a corner of our laboratory. In the corner are two small desks, on the left desk there is a workstation with display, peripheral equipment and a telephone, underneath a drawer unit. In some datasets a wall with vertical blinds and a heating unit was perceived on the left side. On the right side a large box and a part of a blackboard can be seen. We used a middle size robot with a differential drive and caster. On an aluminum frame a slightly opened laptop, a battery and on the top an open box is transported. In our experiments, the robot size was enough to reconstruct the robot, so we added the carton to investigate difficult situations. In addition the robot drove slowly since fast objects are a lesser evil. Fast objects cause early outliers in the registration process and consequently they do not largely disturb the tracking of the background. The initialization as a new object is simple, too. Moreover, our datasets are short and the robot moves from the very beginning. This leads to low weights in the background voxel grid when the robot is already moving and hence the background model is adapted to the moving object. This constitutes the challenge to extract the object before the background model is cluttered with misleading information and maybe a registration failure occur. In summary, this evaluation focuses on situations which are difficult for the initialization, tracking and growing process.

Table 1: Comparison of the original matching strategy with vertex map to model matching proposed in subsection 3.1. All framerates consider exclusively the pure KinFu algorithm, they include the runtimes of the raycaster but not the effort for visualization and grabbing.

dataset	RMSE/ATE [cm]		fps	
	original	ours	original	ours
fr1/xyz	2.10	2.02	58.4	72.3
fr2/xyz	2.23	2.32	54.7	66.2
fr2/desk	9.54	7.88	51.1	58.4

The evaluation was performed with an Intel i7-2600@3.40GHz and a NVIDIA GeForce GTX970. A video of the evaluation results is available at http://www.is.uni-due.de/kinfu_mot/.

4.1 Vertex Map to Model Matching

We evaluated the vertex map to model matching with three datasets from (Sturm et al., 2012). All datasets were obtained from a Microsoft Kinect. The datasets from (Sturm et al., 2012) include highly accurate and time-synchronized ground truth camera poses from a motion capture system, which works with at least 100 Hz and up to 300 Hz. We used the datasets fr1/xyz with 798, fr2/xyz with 3666 and fr2/desk with 2964 depth frames. All three contain an office scene. In fr2/desk the Kinect is moved around two tables so that the loop is closed. To make our and the original approach comparable we used the original PCL implementation as a basis and we modified it as less as necessary. For this subsection we just replaced the matching part and we removed the multi-scale feature without further changes. In particular the algorithm works from a global perspective synchronous and thus the runtimes are comparable, too. We neither adapt the parameters, 10cm are used as ICP distance threshold and the truncation distance is 3cm although a shorter truncation leads to better results. Table 1 denotes the root mean square (RMS) absolute trajectory errors (ATE) and the framerates. The ATE directly measures the difference between points of the true and the estimated trajectory at each frame and it is a good measure to rate SLAM methods. We applied our approach with seven ICP iterations and we did not switch off the visualization and accordingly the raycaster. Because our approach does not require ray casting the framerates could be increased further.

The raycaster takes for fr2/desk an average of 2.86ms with a standard deviation of 1.05ms. Nonetheless, our method shows better framerates in general due to the reduction of the iteration number. The resulting errors depend obviously on the dataset (two improvements and one degradation) but they are similar overall. In summary, we have presented a good alternative matching method with better runtime and we expect particular advantages for our KinFu extensions.

4.2 Sliding Reduction

In this subsection a dataset was used in which the robot drives towards the camera, at the beginning. The robot then turns to the left side (seen from the camera). To make it harder, the camera was pivoted to left and right permanently. The scene is a subject to sliding because at the beginning just an almost planar surface is responsible for the most outliers. The moving object is initialized after 97 frames in the shape of this planar surface. It is found in the area of the box und it grows downwards to the robot. The transformations are underconstrained for about 100 frames until the robot starts to rotate. As shown in Figure 2 the robot can be reconstructed correctly from two sides. After about 700 frames we get two separate models with high quality. In another experiment we deactivated the estimation of the registration stability based on the eigenvalues. Obviously, the assumed position of the object tilts to the right side in Figure 3, after jumping back and forth. Even a 180° rotation about the surface normal is performed (not in the images). The first step of the sliding reducing, the registration with the previous depth map, was still used. Without any sliding reduction approach the tracking diverges after a few frames because of the continuous moving of the camera.

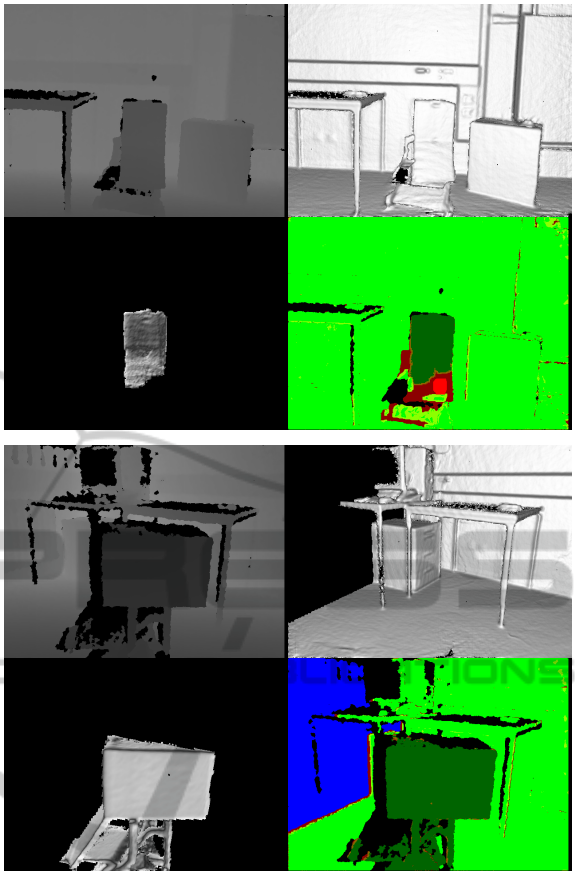

Figure 2: This dataset benefits from sliding reduction. Shown are the results after 131 frames (top) and 711 frames (bottom). The figure represents the depth image, the rendered background, the rendered object and the visualization of the correspondence map. This shows the matched points with the background (light green), the matched points with the object (dark green), potential outlier (yellow), ICP outlier (dark red), potential new object (light red), missing points in depth image (black) and missing correspondences due to volume boundary (blue).

4.3 Background Model Preserving

The registrations of the background model in Figure 4 are not sufficient restricted. One direction is constrained by the wall and another direction is constrained by the floor. However, the whole background voxel grid can be moved to the left or right along the wall and the floor without effecting the ICP error significantly. Only the box on the ground (top image) and later the drawer unit hold the background voxel grid at its position. On the other side the robot moves and turns to the left direction. The robot is a large object with a lot of correspondences which are assigned to the background at the beginning. The robot takes the background voxel grid along with itself, the voxel grid moves to the left. This can be seen after 182

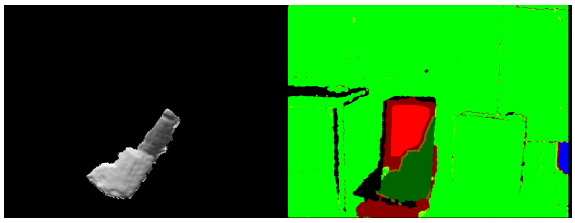

Figure 3: Experiment without the second step of the sliding reduction. The object tilts to the right side.

frames (top) due to the outlier on the right side of the box, the table and the table-leg. The original KinFu cannot handle this dataset, the voxel grid slides further and further until the other wall on the left side is reached. Our algorithm initializes the new object after 162 frames. It needs just a few frames to grow and

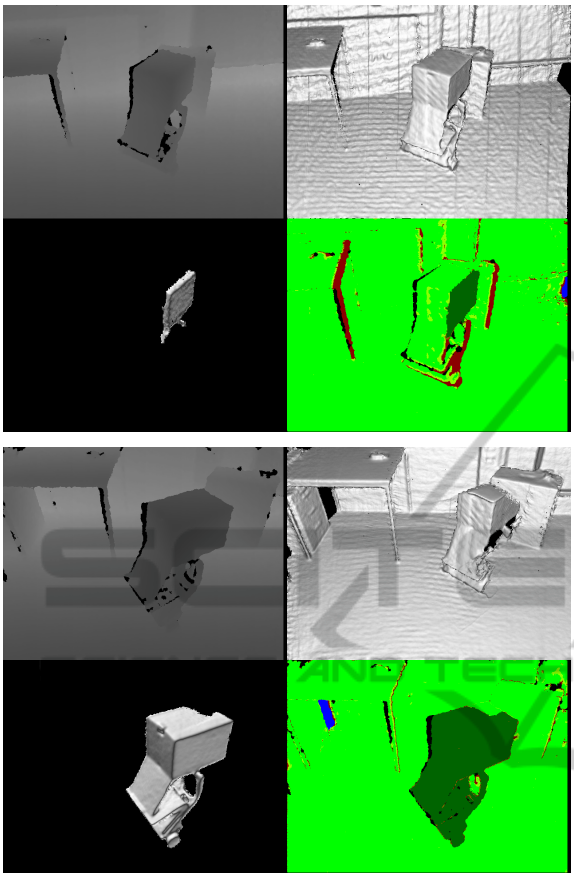

Figure 4: In this dataset the transformation of the background is underconstrained. Shown are the results after 182 frames (top) and 425 frames (bottom). A huge part of the robot is still in the background model at this moment (bottom) since the view to the empty space was blocked by the robot and the voxels there remain without update. The background model becomes clear after the view is free, 300 frames later.

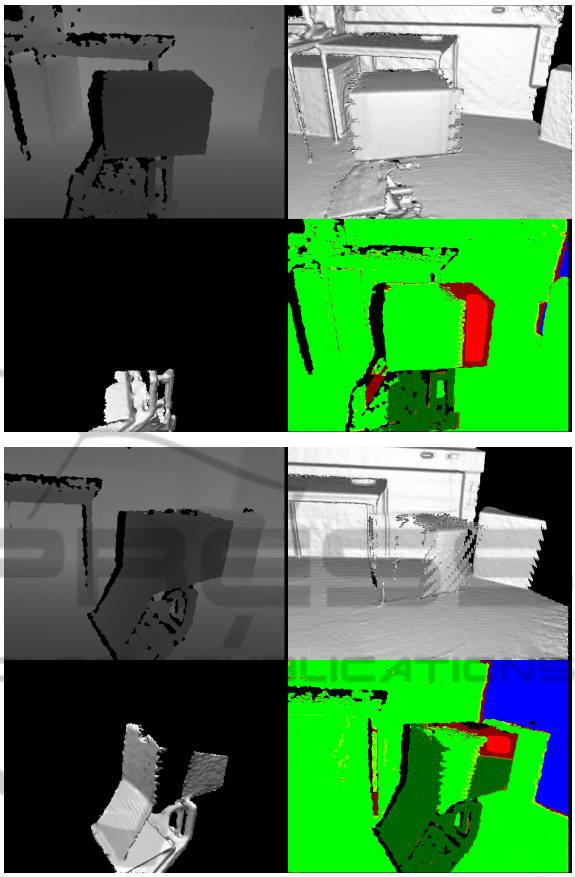

Figure 5: This dataset shows a parallel moving object. The box can be matched with the background for a long time. Long enough to move within the background model (Different positions of the box within both rendered background images.). Shown are the results after 343 frames (top) and 613 frames (bottom).

after a major part of the correspondences is assigned correctly the position of the background voxel grid remains stable. Actually, it returns a little bit back to the original position. A complete return is impossible because the background model is already filled with shifted data. If the robot would not move from the beginning the weights of the background model could rise and the model would become more stable. After the object initialization, the background voxel grid could return to the old position.

4.4 Parallel Moving Object

An object which moves almost parallel to the image plane can be a special challenge. Some parts produce outliers during other parts can be matched to the background for a long time. In addition the background voxel grid gets the change to assimilate data from the moved object. This also leads to a late initialization. In Figure 5 the object is initialized after 209 frames. This time the growing starts at the robot and not at the box. By the way, this example shows the capability to reconstruct small objects. The size of the first voxel grid was $43 \times 53 \times 60$ voxels ($25.2 \times 31.1 \times 35.2$cm) and the size shown in the top of Figure 5 was reached 133 frames and 15 grid growing steps after the initialization. Due to the distance between robot framework and the box it takes very long before the model reaches the whole box. As shown in the bottom image a new potential object is already detected. The initialization delays which are triggered by grid growing are very useful in this situation.

4.5 Current Investigations

The theoretical framerate regarding our datasets, con-

sidering just the pure gpu processing pipeline (comparable to subsection 4.1), is about 40fps with one reconstructed object beside the background. The framerate including five windows and grabbing during the reconstruction of one object is around 25fps (in comparison with 12.6fps of the original KinFu from PCL). The framerate is limited by the painting of the windows. The complete algorithm with one object and deactivated visualization runs with more than 30fps.

If three objects are initialized the reduction of the framerate is negligible. This is due to the fact that the bottleneck is the communication and synchronization between CPU and GPU. Accordingly, the computer hardware is insufficient utilized. Since we use GPU streams, more objects just mean a better utilization.

We did some experiments with tracking of humans as example for non-rigid objects, too. We get good results if we allow just one moving object. This observation conforms to (Izadi et al., 2011). Without object number limit several parts of the body get a separate voxel grid. The separation is reasonable (individual movement of the extremities) but some parts become relative small and may get occluded during the movement. Since our algorithm does not consider relationships between the tracked objects this is not allowed at the moment. However, as long occlusion is avoid and the body parts do not get too small the results are stable.

5 CONCLUSIONS

We extended KinectFusion by the ability to track and reconstruct several moving objects simultaneously. We propose an alternative matching strategy and some further modifications to the GPU processing pipeline. The capabilities of our system are demonstrated with three examples. It was shown, that the stability of object tracking is enhanced due to sliding reduction. Furthermore, the robustness of the determination of the camera poses in scenes with moving objects is improved. Finally, we give an example in which, at the beginning a small object can be tracked during its movement parallel to the image plane. Despite the new functionalities our systems is still real-time capable.

For future work, we plan to focus on the initialization. First, we would like to move data from the background voxel grid to the new object voxel grid during the initialization. Second, the detection of moving object pixels during the initialization should rely on a complete pixel based segmentation of the correspondence map. This would enhance the completeness of the object initialization and allow to relax the constrains for triggering a new initialization. Beyond this, we intend to consider the relations between the objects with technics from the field of articulated objects. This can further improve the registration results – especially of small independent moved object parts – and enable us to implement a voxel grid merge mechanism for objects which were wrongly initialized twice.

REFERENCES

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA. ACM.

Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W. (2012). An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, St. Paul, MA, USA.

Gelfand, N., Ikemoto, L., Rusinkiewicz, S., and Levoy, M. (2003). Geometrically stable sampling for the ICP algorithm. In *Fourth International Conference on 3D Digital Imaging and Modeling (3DIM)*.

Heredia, F. and Favie, R. (2012). Kinfu Large Scale in PCL. http://www.pointclouds.org/blog/srcs/fheredia/.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. (2011). Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. ACM Symposium on User Interface Software and Technology.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *IEEE ISMAR*. IEEE.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *International Conference on Robotics and Automation*, Shanghai, China.

Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*.

Whelan, T., Kaess, M., Fallon, M., Johannsson, H., Leonard, J., and McDonald, J. (2012). Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia.