Photorealistic 3D Mapping of Indoors by RGB-D Scanning Process

Tommy Tykkala, Andrew I. Comport, Joni-Kristian Kamarainen

Photorealistic 3D Mapping of Indoors by RGB-D Scanning Process

Tommi Tykkälä[1], Andrew I. Comport[2], and Joni-Kristian Kämäräinen[3]

***Abstract*— In this work, a RGB-D input stream is utilized for GPU-boosted 3D reconstruction of textured indoor environments. The goal is to develop a process which produces standard 3D models from indoors to explore them virtually. Camera motion is tracked in 3D space by registering the current view with a reference view. Depending on the trajectory shape, the reference is either fetched from a concurrently built keyframe model or from a previous RGB-D measurement. Real-time tracking (30Hz) is executed on a low-end GPU, which is possible because structural data is not fused concurrently. After camera poses have been estimated, both trajectory and structure are refined in post-processing. The global point cloud is compressed into a watertight polygon mesh by using Poisson reconstruction method. The Poisson method is well-suited, because it compressed the raw data without introducing multiple geometries and also fills holes efficiently. Holes are typically introduced at occluded regions. Texturing is generated by backprojecting the nearest RGB image onto the mesh. The final model is stored in a standard 3D model format to allow easy user exploration and navigation in virtual 3D environment.**

I. INTRODUCTION

In this work, a process model is developed to reconstruct watertight and textured 3D maps from indoor environments by using a RGB-D sensor. The models are stored in a standard format to allow further use, such as virtual exploration, robot teleoperation and 3D printing. Most of the current visual odometry and mapping techniques do not consider model re-distribution and visual quality, because more coarse representations can be used in navigation tasks. In our workflow, photorealistic 3D models are produced by using a laptop with a low-end GPU and a RGB-D sensor.

In our previous work, we have concentrated on real-time augmented reality in a studio environment [1]. To prevent camera tracking drift and to allow foreground actors, a static keyframe model was used as reference. However, several application domains exist where the digitized 3D model can be directly valuable. In robotics, tele-operation tasks benefit from a 3D model. Websites support online 3D product catalogs and 3D printing devices are becoming available for consumer market. Also apartment renovation and re-organization can benefit from a realistic 3D model. Currently there is no standard process, which would produce visually pleasing 3D content for further use using commodity hardware. The available systems store the content in internal formats such as a point cloud (Figure 1), a Gaussian mixture model or a voxel grid on a GPU, which are not directly usable in standard softwares. The final memory consumption may also become unnecessarily high due to naïve reconstruction methods which introduce multiple surface instances.

Fig. 1. `Room B` sequence. A raw point based model generated using the SLAM mode. The apartment is scanned using a RGB-D sensor by hand in a single 360° sweep. The result is not a photorealistic 3D model and memory consumption becomes high.

Our contributions are the following. 1) A scanning process is developed from raw RGB-D measurement stream into a textured and watertight 3D model, which is stored compactly in a standard Wavefront format. 2) A photometrical refinement phase is developed, which improves the model keyframe quality by fusing hundreds of RGB-D measurements together. The sweeping process produces all necessary data for generating re-usable photorealistic 3D models. The core function is the camera tracking module which operates in real-time on a low-end GPU (NVS4200m with 48 CUDA cores) [1]. Watertight and textured polygon mesh is generated in an automatic post-process. The model is stored in Wavefront format, which also converts also to WRML, and therefore enables online 3D visualizations.

II. PREVIOUS WORK

Early visual Simultaneous Localisation and Mapping (vSLAM) methods represent environment as a sparse set of 3D points which is used as a fixed reference for camera tracking. The typical process is to detect and track a sparse set of feature points which are matched in several images. The first feature-based visual SLAM methods were based on the extended Kalman filter (EKF) [3]. However, local bundle adjustment replaced EKF, because it is more accurate and still remains real-time. PTAM [4] separated tracking and mapping into two parallel modules, where the mapping

[1] Machine Vision and Pattern Recognition Laboratory, Lappeenranta University of Technology, Kouvola Unit, Finland

[2] CNRS-I3S/University of Nice Sophia-Antipolis, France

[3] Department of Signal Processing, Tampere University of Technology, Finland

part was essentially bundle adjustment. At the same time, Comport *et al.* developed a novel dense image registration technique for urban pose estimation using a calibrated stereo camera [13]. DTAM was introduced as the dense version of PTAM, which allowed dense tracking and mapping using a monocular camera [5]. Due to significant indoor RGB-D sensor developments, several systems have abandoned stereo cameras. Engelhard *et al.* based their RGB-D tracking on traditional feature extraction and matching but with additional ICP and graph optimization phases [16]. Audras *et al.* implemented RGB-D camera tracking as a direct photometric error minimization without feature extraction [12]. The system was executed in real-time on a powerful desktop CPU. Also Newcombe replaced monocular camera by the Kinect sensor in the KinectFusion system [6], which generates dense reconstructions of small objects and spaces. In KinectFusion, the camera is tracked using the standard ICP algorithm, which is executed on a GPU. The system also integrates point measurements to a voxel-based 3D model concurrently. Unfortunately the system does not scale into larger operating volumes, because a dense voxel grid is not memory-wise scalable [1]. In Kintinuous system, the memory scalability problem is addressed by converting the reconstruction volume into triangles whenever it has to be moved [15]. Henry integrate measurements into a surfel map [14]. Also Stückler uses surfel map during pose estimation and store it using a sparse octree representation to better avoid memory limitations [7]. In our previous work, RGB-D measurements are integrated into a small number of keyframe views [1]. Data fusion may use larger resolution to avoid introducing averaging or discretization errors [17].

Many application domains, such as 3D visualization and printing, benefit precise models. Despite that relatively accurate RGB-D pose estimation and mapping approaches exist, most of the methods do not automatically output photorealistic 3D models which are compatible with standard modeling softwares. Whelan, in fact, is the only one to suggest a specific technique [11] to generate textured polygon meshes using RGB-D mapping process [15]. We presented a real-time GPU implementation of a RGB-D camera tracker [1], for which we now develop a process to produce photorealistic 3D models.

III. THE PROCESS MODEL

After recording a video, GPU-boosted RGB-D tracking is executed which produces 3D trajectory [1]. Whether or not RGB-D tracking uses keyframes, only the trajectory is stored. The model keyframes are selected by looping the trajectory and storing a keyframe whenever user-specified angular or translational distance to the existing model is exceeded. The neighboring RGB-D measurements to the keyframes are efficiently localized (timestamp or frame index) and depth map fusion is executed. In depth map fusion, keyframe depth maps are filtered using all RGB-D measurements available. Then, optionally, bundle adjustment is possible for the full model. The results presented in this paper do not use bundle adjustment at all, because the sequences are relatively short and pose error remains small. Finally watertight polygon model is generated from the RGB point cloud using Poisson reconstruction method. Keyframe images are stored into a single texture and UV coordinates are generated for each polygon. The textured mesh is then stored in a simple Wavefront format which can be loaded into various standard 3D modeling programs for further refinement. The video[1] illustrates the full process for sequence `Room A`.

The phases in our process are thus

1) Record RGB-D video (manual)
2) Generate 3D trajectory by RGB-D tracking (automatic)
3) Select keyframes (automatic)
4) Depth map fusion (automatic)
5) Optional : bundle adjustment (semi-automatic)
6) Watertight polygonization (automatic)
7) Texture map generation (automatic)
8) UV coordinate generation (automatic)
9) Store Wavefront mesh (automatic)

IV. PHOTOMETRICAL COST FUNCTION

The pose estimation is defined as a direct image registration task between the current image $\mathbf{I}:\mathbb{R}^2 \Rightarrow \mathbb{R}$ and the nearest keyframe $\mathbf{K} = \{\mathbf{P}^*, \mathbf{c}^*, \mathbf{T}^*\}$, where $\mathbf{P}^* = \{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n\}$ is a set of 3D points and $\mathbf{c}^* = \{c_1, c_2, \ldots, c_n\}$ are the corresponding color intensities. The 4×4 matrix $\mathbf{T}^*$ defines the keyframe pose relative to the reference coordinate system. $\mathbf{T}^*$ maps points from the reference to the keyframe coordinate system. $*$ is used to denote fixed reference variables.

The task is to find the correct pose increment $\mathbf{T}(\mathbf{x})$, that minimizes scalar cost function $\frac{1}{2}\mathbf{e}(\mathbf{x})^T\mathbf{e}(\mathbf{x})$, where the residual is defined by

$$\mathbf{e} = \mathbf{I}\left(w\left(\mathbf{P}^*; \widehat{\mathbf{T}}\mathbf{T}(\mathbf{x})\right)\right) - \mathbf{c}^*. \tag{1}$$

Parameters $\mathbf{x} \in \mathbb{R}^6$ produce $\mathbf{T}(\mathbf{x}) \in \mathbb{SE}(3)$ motion when applying a Lie generator, $\widehat{\mathbf{T}}$ is the base transform and $w(\mathbf{P};\mathbf{T})$ is the warping function which transforms and projects 3D points by

$$w(\mathbf{P};\mathbf{T}) = \mathbf{K}D(N(\mathbf{R}\mathbf{P}+\mathbf{t}), \alpha), \tag{2}$$

where $\mathbf{T} = \begin{bmatrix}\mathbf{R} & \mathbf{t}\\ 0 & 1\end{bmatrix}$, $N(\mathbf{p}) = (p_1/p_3, p_2/p_3)$ dehomogenizes a point, $D(\mathbf{p}, \alpha)$ performs lens distortion using standard Caltech parameters $\alpha \in \mathbb{R}^5$ (constant) and $\mathbf{K}$ is 3×3 intrinsic matrix of the camera (constant).

Temporal correspondence problem between feature points fully avoided, and the warping function will produce the dense correspondence after the correct pose is found. The estimation is robust due to gradient-based pixel selection and a M-estimator. Inverse compositional approach is used for minimization, which allows pre-computing the Jacobian [1][8]. Minimization enables precise camera tracking. Accuracy and comparison with KinectFusion were documented in our earlier work [1].

[1] `http://youtu.be/tD3lFxrCHaw`

V. RGB-D TRACKING MODES

RGB-D tracking can be executed in three different modes: SLAM (algorithm 1), incremental (algorithm 2), and keyframe mode. The main difference is in selecting the reference keyframe for tracking. In SLAM mode, the keyframe database is built concurrently while tracking the camera. Loop closures are maneuvered whenever re-visiting scenes that are already mapped. In incremental mode, a recent RGB-D measurement is taken as a reference, and memory consumption remains extremely small. In keyframe mode, the reference is selected as the nearest neighbor in a pre-recorded database. Keyframe mode is only usable after an environment has been mapped.

In SLAM mode, the first keyframe is generated with the identity pose using the first RGB-D measurement. Tracking starts relative to the first keyframe, and new keyframes are added when no keyframe is not found within a narrow search domain. In this case, a larger search domain is used to guarantee that at least one keyframe is found. A pose search domains are thus expressed as angle and distance threshold pairs $(\theta_{\max}, \mathbf{d}_{\max})^{\{1,2\}}$. The nearest keyframe in a search domain, is the one which has the smallest pose distance. The distance metric was defined in our earlier work [1] and it unifies rotation and translation errors into a scalar value

$$s = \underset{k \in \Omega}{\operatorname{argmin}} \; ||w(\mathbf{P}; \Delta\mathbf{T}_k) - w(\mathbf{P}; \mathbf{I})||, \qquad (3)$$

where Ω is a set of potential candidates, $\Delta\mathbf{T}$ is a relative transformation between the frames, and $\mathbf{K}_s$ is the nearest keyframe. The test point set $\mathbf{P}$ is a sparse representation of the view frustum. In particular, the frustum is approximated by n sparse 2D grids, each having uniformly sampled depth coordinates in the overall depth range of the RGB-D sensor.

In the current implementation, keyframes can be added until the limit of GPU memory is reached. Keyframes are compressed by a point selection procedure, where only a fraction of the RGB-D points need to be stored based on the image gradient magnitude. SLAM mode is useful especially when the camera is known to re-enter scenes. For example, when building a 3D model of an apartment, it is sometimes necessary to add missing keyframes to remove holes in the model. `Kitchen` sequence demonstrates SLAM mode use case (Fig. 2, and video[2]). The downside of SLAM mode is that in practise keyframe density in 3D space becomes smaller than in the incremental mode, where baseline is only few frames. The previous frame is often the best reference, because Lambertian assumption holds well and large occlusions do not exist. In `Room B` sequence, the trajectory is a direct 360° where camera does not re-visit same views except at the end (see video[3]). Incremental tracking produces more accurate result (Fig. 3). The reference update is performed in each frame. By selecting a fit tracking mode based on camera trajectory, pose estimation bias becomes small and the global optimization can be avoided.

[2] `http://youtu.be/aFrVROLja38`
[3] `http://youtu.be/mfc1KV7DDpI`

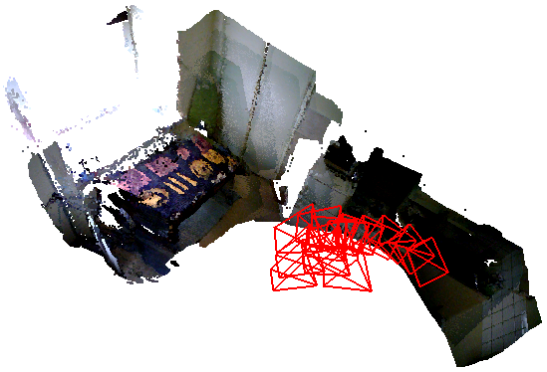


Fig. 2. `Kitchen` sequence. A raw point based model is generated using the SLAM mode. The camera re-visits same keyframes many times, and the errors cumulated during the loop are avoided[3].

Algorithm 1 SLAM mode algorithm.

Require: Keyframe database contains 1st RGB-D measurement
Input: $\mathbf{T}_{cur} = \mathbf{I}$, Search domains $\{\theta^1_{\max}, \mathbf{d}^1_{\max}\} \leq \{\theta^2_{\max}, \mathbf{d}^2_{\max}\}$
Output: Trajectory $\{\mathbf{I}, \mathbf{T}^1_{cur}, \ldots, \mathbf{T}^n_{cur}\}$, Keyframe model $\mathbf{K}$.

for each RGB-D measurement **do**
  newKeyFlag = false
  $\mathbf{F} = \{\mathbf{P}^*, \mathbf{c}^*, \mathbf{T}_{key}\} \Leftarrow$ FindKeyframe$(\mathbf{T}_{cur}, \theta^1_{\max}, \mathbf{d}^1_{\max})$
  if $\mathbf{F} = \emptyset$ **then**
    newKeyFlag = true
    $\{\mathbf{P}^*, \mathbf{c}^*, \mathbf{T}_{key}\} \Leftarrow$ FindKeyframe$(\mathbf{T}_{cur}, \theta^2_{\max}, \mathbf{d}^2_{\max})$
  end if
  $\widehat{\mathbf{T}} \Leftarrow$ Minimize$(\mathbf{T}_{cur}\mathbf{T}^{-1}_{key}, \mathbf{P}^*, \mathbf{c}^*, \mathbf{I}_{cur})$ (eq. 1)
  $\mathbf{T}_{cur} \Leftarrow \widehat{\mathbf{T}}\mathbf{T}_{key}$
  if newKeyFlag = true **then**
    $\{\mathbf{P}, \mathbf{c}\} \Leftarrow$ Select a subset of points with largest gradients
    Precompute Jacobian
    Add keyframe to model : $\mathbf{K} = \{\mathbf{K}, \{\mathbf{P}, \mathbf{c}, \mathbf{T}_{cur}\}\}$
  end if
end for

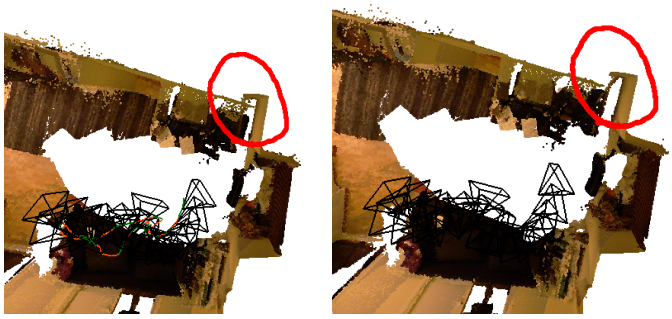
VI. DEPTH MAP FUSION USING RGB-D DATA

Because the keyframes are sparsely selected from the stream of RGB-D measurements, they do not utilize all available information. The intermediate point clouds, which are not selected as keyframes, are warped into a nearest keyframe and the final maps are filtered in post-processing. In effect, this improves depth map accuracy and fills holes. A method is required for determining the correct mode, because the warped depths may also produce a multi-modal distributions in case of occlusions. We assume the correct mode is near the median of the depth values. Note that area covered by a pixel grows as a function of distance. This should be taken into account by avoiding points which are too far away from the ray through pixel center point (Figure 4). A large portion of the outlier points can be neglected based on color deviation to the keyframe pixels. Thus, prior to any filtering, it is useful to discard the points whose color difference to the reference color is larger than a threshold. If the number of remaining points is small, the depth value should not be filtered, but completely discarded as noise. The median depth is computed from the remaining sample points, which determines a distance range around a surface. The

Algorithm 2 Incremental mode algorithm

Require: $\{\mathbf{P}^*, \mathbf{c}^*\} \Leftarrow$Select the best points from 1st RGB-D.
Input: $\mathbf{T}_{cur} = \mathbf{T}_{ref} = \mathbf{I}$
Output: Trajectory $\{\mathbf{I}, \mathbf{T}^1_{cur}, \ldots, \mathbf{T}^n_{cur}\}$.

```
for each RGB-D measurement do
    T̂ ⇐ Minimize(I, P*, c*, I_cur) (eq. 1)
    T_cur ⇐ T̂ T_ref
    if reference update signaled then
        {P*, c*} ⇐ Select a subset of points with largest gradients
        Precompute Jacobian
        T_ref = T_cur
    end if
end for
```



(a) (b)

Fig. 3. `Room B` sequence. Reconstruction bias in when operating in a) incremental mode, b) SLAM mode. After 360° turn the first and last keyframe should map points consistently into a single 90° corner. In this case a) is slightly more precise, because subsequent images are photometrically the most comparable with negligible interpolation errors.

samples near the surface are locally averaged within initial bounds and standard deviations σ_k are computed. In practise, OpenMP is used to parallelize computations for depths.

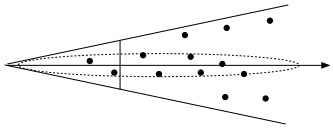


Fig. 4. The area covered by a pixel becomes larger at longer distances. The pixel median can be improved by focusing on the samples near a ray through the pixel center.

The depth maps can be refined further by minimizing photometrical criteria. The necessary pre-condition is that the precise depth values are known to reside within bounded ranges, and image measurements with relatively accurate camera poses are available. From the previous phase, standard deviations σ_k are indeed available and a population of depth candidates can be generated. The depth range $[z_m - \delta, z_m + \delta]$ is sampled evenly to enumerate the solution candidates $\mathbf{P}_k$ which are then projected into surrounding images for cost evaluation (Figure 5). The optimal depth z_o is selected using a photometrical error function

$$z_o = \mathbf{e}_3^T \left(\operatorname*{argmin}_k \sum_{j=1}^{n} \left(\mathbf{I}^j(w(\mathbf{P}_k; \mathbf{T}_j)) - \mathbf{I}^*(w(\mathbf{P}_k; \mathbf{I})) \right)^2 \right), \quad (4)$$

where $\mathbf{e}_3^T = (0,0,1)^T$ selects the z-coordinate of solution point $\mathbf{P}_o$. It is noted that this cost function will provide arbitrary results for points which are on homogeneous image regions without any gradient. Therefore photometrical refinement can only be done for the points with sufficient image gradient magnitude. In DTAM, this problem was addressed by assuming smoothness when the image gradient has small magnitude [5]. The benefit with discrete optimization is that it works even when the amount of image measurements is small. In our implementation, the refinement is not done in case the optimal depth is found at the search range boundary. This is to ensure that a local minimum is within the range.

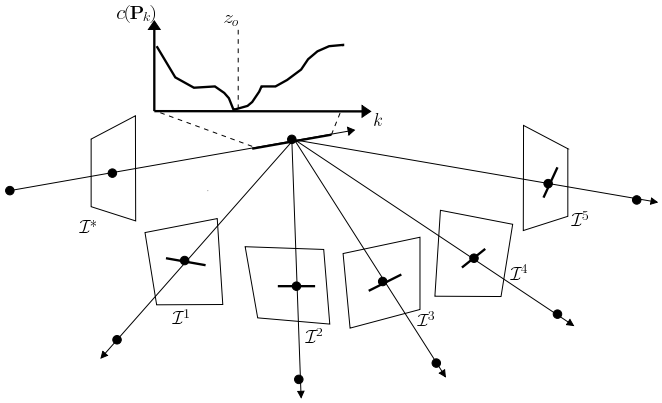


Fig. 5. Photometrical refinement using bounded depth range. The optimal depth has the most similar color in all images. The pose configuration is assumed to be precise.

VII. WATERTIGHT POLYGONIZATION

Polygon models are compact in their memory consumption and are better supported by standard 3D modeling programs than point clouds. A polygonization phase generates a polygon mesh from a point cloud. In our context, the method should take into account noise and missing data. A common approach is to fit the points to a surface using the zero-set of an implicit function, such as a sum of radial bases or piecewise polynomial functions. We select the Poisson method, because it produces a watertight surface based on a photometrically refined, oriented point cloud [2]. The point normals are derived from the depth fused maps. A depth map is converted into a point cloud by

$$\mathbf{P}(u,v) = \mathbf{T}_b \begin{bmatrix} z(u,v)\mathbf{K}_{IR}^{-1}(u \quad v \quad 1)^T \\ 1 \end{bmatrix}, \quad (5)$$

where $z(u,v) : \mathbb{R}^2 \Rightarrow \mathbb{R}$ is the depth map, $\mathbf{T}_b$ is the 4×4 base transform between IR and RGB view and $\mathbf{K}_{IR}$ is the 3×3 intrinsic matrix of IR view.

A normal map can then be defined as

$$\mathbf{n}(u,v) = \Big(\mathbf{P}(u+1,v) - \mathbf{P}(u,v)\Big) \times \Big(\mathbf{P}(u,v+1) - \mathbf{P}(u,v)\Big). \quad (6)$$

Unit normals are generated by $\mathbf{n}(u,v) \Leftarrow \mathbf{n}(u,v)/\|\mathbf{n}(u,v)\|$.

Oriented points are transformed into a reference coordinate system. The Poisson method finds a scalar function whose gradients best match the vector field, and extracts

the appropriate isosurface. The algorithm uses OpenMP for multi-threaded parallelization and octree data structure to reduce memory consumption [2]. To further avoid memory limitations, the mesh could also be done piece-by-piece using a single, moving reconstruction volume. The most interesting parameters are octree grid resolution, point weights and minimum point count in an octree node. Because octree resolution is limited, the resulting mesh may become over-smooth at complex regions. The reconstruction accuracy depends mostly on the precision of the oriented point cloud. With Microsoft Kinect sensor, the depth noise increases with distance. In our tests, we measure distance to the camera and set quadratically decaying point weights. Poisson reconstruction result without and with depth fusion can be observed in Figure 6. Notice how Poisson method generates the floor and fills holes despite that measurements do not exist (compare with Fig. 1).

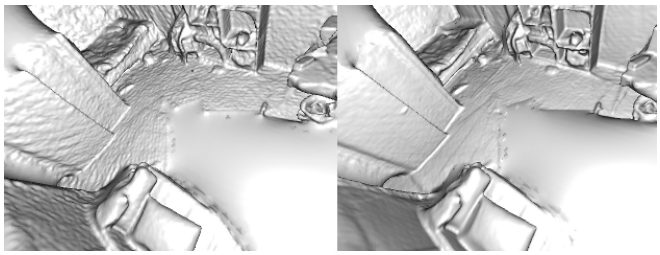


Fig. 6. `Room B` sequence. Watertight Poisson reconstruction without and with depth fusion. Notice how holes are filled and missing regions such as the floor appears.

VIII. MESH TEXTURING

The example sweep in Figure 1 created 23 RGB-D keyframes whose textures are redundant in color due to Bayer filtering. The images are downsampled into 320×240 resolution and stored into a 2048×2048 texture (Figure 7). The size is good for testing purposes and it allows 6×8 keyframes to be used.

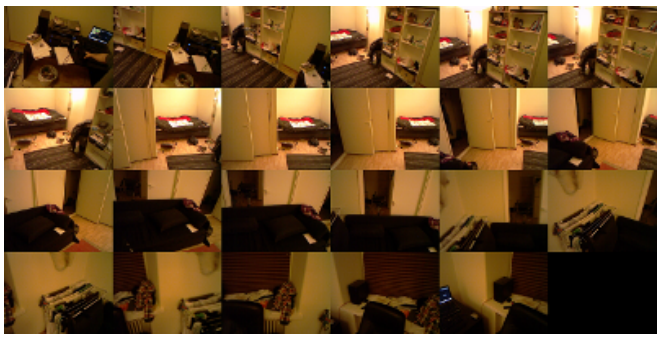


Fig. 7. Keyframe images are stored into a single big texture.

The Poisson polygons which are in the visible range of the RGB-D sensor are projected onto all keyframe views and UV-coordinates are generated. Poisson reconstruction module outputs polygons with n corner points. 2D area of a polygon $\triangle$ is evaluated using the following formula

$$A(\triangle) = \frac{1}{2}\sum_{k=0}^{n} \det\Big(\begin{bmatrix} \mathbf{p}_k^T \\ \mathbf{p}_{\mathrm{mod}(k+1,n)}^T \end{bmatrix} \Big), \tag{7}$$

where points $\mathbf{p}_k = (u,v)^T$ are the corner points of a 2D polygon. The formula applies to convex and concave polygons as long as they are not self-intersecting. When a polygon is visible in more than one view, we choose to favor largest spatial resolution with the formula

$$\triangle_{\mathrm{uvkey}} = \underset{j}{\mathrm{argmax}}\ A_j(\triangle) \in \mathbb{N}, \tag{8}$$

where $\triangle_{\mathrm{uvkey}}$ is the index of the best UV-mapping keyframe (Fig. 8). Because frequent switches in UV-mapping directions can cause visually disturbing seams, mapping can be improved by enforcing the locally dominant keyframe. One method to do so is to recursively enumerate connected polygon neighbors in n passes, and then prefer the mapping direction which has the largest number of votes. Finally the selected UV-coordinates are converted into global texture coordinates and stored. The keyframe images are not undistorted to better maintain maximum texture quality. The resulting meshes can be observed in Figure 9.

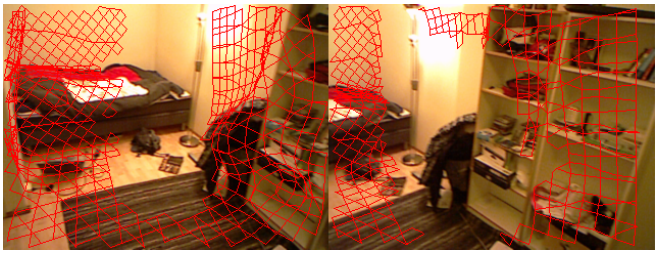


Fig. 8. Poisson(7) polygons are projected onto keyframe images. UV-coordinates can be chosen from the view which has the best spatial resolution. The corner of the shelf is visible in two different keyframes, but the selection favors the left image, because the camera is closer.

When final texturing is patched together from keyframes, some color banding effects may occur if brightness varies across the images. Manual camera settings reduce global lighting variation across images. To reduce the problem further, averaging or median filtering could be attempted. Also more sophisticated Laplacian pyramid filtering has been suggested for the task [18].

The memory consumption is shown in Table I. The consumption is separated into geometry and texture consumption for Poisson meshes. The datasets `Room A`, `Room B` and `Kitchen` have 47, 23 and 14 keyframes. After the trajectory has been estimated, a raw point cloud is generated. The minimum requirement per vertex is 9 attributes (position, normal and color). After Poisson reconstruction and texture mapping, the vertices require only 3 floats (x,y,z) and n triangles have in total $9*n$ attributes ($3*2$ UV coordinates + $3*1$ index). In addition texture map requirement is computed directly as $k*320*240*3$, where k is the number of keyframes. Table I shows memory footprint for $2^7, 2^8$, and 2^9 octree grids. The corresponding geometric quality is

(a)

(b)

Fig. 9. Final textured Poisson meshes loaded into Meshlab for inspection: a) `Room B`, b) `Kitchen`. Poisson reconstruction produces watertight mesh, whose texturing is photorealistic as it is directly mapped from the keyframe images. The cost of reduced memory footprint is over-smoothing, which may occur at thin surfaces such as the shelf in 9a. Also lighting changes can be detected at seams where texture data source switches from one keyframe to another. Otherwise the models are photorealistic and in metric units.

illustrated in Figure 10. The reconstructions in Figure 9 are generated using 2^9 grid.

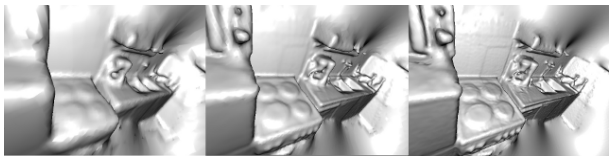


Fig. 10. Kitchen scene reconstructed with 2^7, 2^8 and 2^9 octree resolution. Phong shading reveals the level of geometrical details at each resolution.

IX. CONCLUSIONS

We have shown how photorealistic 3D models can be captured using a RGB-D sensor using a laptop and low-end GPU. The main benefit is in storing the models compactly in a standard format to enable their further use in different applications. Two mapping modes were discussed, which do not always require global optimization. The incremental mode fits trajectories, which have only a small number of potential loop closures. Otherwise the SLAM mode is recommended. A watertight polygon mesh was generated using the Poisson method and mesh appearance was directly mapped from the keyframe images. The resulting models

Dataset	Raw	Poisson(9)	Poisson(8)	Poisson(7)
`Room A`	$124MB$	$(32+12)MB$	$(7.8+12)MB$	$(2.0+12)MB$
`Room B`	$60.6MB$	$(21+5)MB$	$(5.8+5)MB$	$(1.6+5)MB$
`Kitchen`	$36.9MB$	$(14+3)MB$	$(3.8+3)MB$	$(1.1+3)MB$

TABLE I

MEMORY CONSUMPTION OF THE TEST SEQUENCES.

are photorealistic despite that they have significantly lower memory footprint than raw point clouds. All phases in the process are automatically executed after a RGB-D video has been recorded. In the future, our goal is to experiment with 3D printing and develop easy-to-use tools for lighting normalization and global keyframe model refinement.

REFERENCES

[1] T. Tykkala, H. Hartikainen, A.I. Comport and J. Kamarainen, Live RGB-D Camera Tracking for Television Production Studios, Journal of Visual Communication and Image Representation, Elsevier, 2013.

[2] M. Kazhdan and H. Hoppe, Screened Poisson Surface Reconstruction, Transactions on Graphics, 2013.

[3] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse, MonoSLAM: Real-Time Single Camera SLAM. PAMI, 2007, Vol. 4, pp. 1052–1067.

[4] G. Klein, and D. Murray, Parallel Tracking and Mapping for Small AR Workspaces, ISMAR, 2007, pp. 225–234.

[5] R.A. Newcombe, S. Lovegrove, A.J. Davison, and A.J. Miller, DTAM: Dense tracking and mapping in real-time, ICCV, 2011, Vol. 1.

[6] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges and A. Fitzgibbon, KinectFusion: Real-time dense surface mapping and tracking, ISMAR, 2011, pp. 127–136.

[7] J. Stückler and S. Behnke, Integrating Depth and Color Cues for Dense Multi-Resolution Scene Mapping Using RGB-D Cameras, In Proceedings of IEEE International Conference on Multisensor Fusion and Information Integration (MFI).

[8] S. Baker and I. Matthews, Lucas-Kanade 20 Years On: A Unifying Framework, Int. J. Comput. Vision, 2004, Vol. 56, Number 3, pp. 221–255.

[9] C. Herrera, J. Kannala and J. Heikkila, Joint depth and color camera calibration with distortion correction, PAMI, 2012, Vol. 34, 10.

[10] M.I. A. Lourakis and A.A. Argyros, SBA: A Software Package for Generic Sparse Bundle Adjustment, ACM Trans. Math. Software, 2009, Vol. 36, 1, pp. 1–30.

[11] Z.C. Marton, R.B. Rusu and M. Beetz, On fast surface reconstruction methods for large and noisy point clouds, ICRA, 2009, pp. 3218–3223.

[12] C. Audras, A.I. Comport, M. Meilland, and P. Rives, Real-time dense RGB-D localisation and mapping, Australian Conference on Robotics and Automation, 2011

[13] A.I. Comport, Accurate Quadri-focal Tracking for Robust 3D Visual Odometry, IEEE Int. Conf. on Robotics and Automation (ICRA), 2007

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

[15] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard, Kintinuous: Spatially extended KinectFusion, in RSS

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

[17] M. Meilland and A.I. Comport, Super-resolution 3D Tracking and Mapping, IEEE International Conference on Robotics and Automation, 2013.

[18] R. Szeliski, Image Alignment and Stitching, Foundations and Trends in Computer Graphics and Vision, Vol 2, Number 1, 2006.