3D Scene Reconstruction from RGB-Depth Images

JIANWEI ZHANG*, FANWEI KONG*, and CHENGJIE ZHU, UC Berkeley

Fig. 1. Visualization of the inputs and reconstruction results of our 3D scene reconstruction framework.

3D scene reconstruction from multiple RGB-D images has become a popular research topic, as it can be applied in many emerging techniques, such as augmented virtual reality, gaming and robotics. Obtaining a reconstructed 3D model of a scene at high quality is thought to be a challenging task due to the difficulties in fusing noisy depth data into reliable 3D representation, artifact corrections, and building efficient and scalable algorithms to complete the task within a reasonable computing time. In contrast to existing complex 3D scene reconstruction systems, we have built a simple system based on concepts that we can easily understand. Our system can reconstruct dense mesh of a room-scale environment. The system comprises two main parts: camera pose estimation and mesh reconstruction. We applied visual odometry to estimate camera pose at each frame to merge RGB-D images into a truncated signed distance function (TSDF) volume under a consistent global coordinate system. We then reconstructed meshes from the TSDF volume using marching cube. .

ACM Reference Format:
Jianwei Zhang*, Fanwei Kong*, and Chengjie Zhu. 2020. 3D Scene Reconstruction from RGB-Depth Images. *ACM Trans. Graph.* 37, 4, Article 111 (August 2020), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 TECHNICAL APPROACH

Existing 3D scene reconstruction systems usually use a combination of complex algorithms to construct high quality meshes from noisy depth data [Dai et al. 2017a,b]. Although those systems have achieved significant progress in artifact correction and constructing detailed features within a short amount of time, their systems can be challenging for other people to understand and study due to the complexity. Therefore, we decided to build a relatively simple 3D scene reconstruction system based on concepts that we can easily understand.

Our reconstruction framework requires inputs of the color and depths images. We took the color and depth images using an Intel®Realsense ™depth camera. Using images from each frame, our framework first estimates the camera poses using visual odometry and fuse the RGB-D images using TSDF. We can optionally construct the mesh of the scene at each frame or at the last frame using the marching cube algorithm.

1.1 Visual odometry

We built a simple visual odometry system and our implementation is inspired by the Visual SLAM tutorial [Xiang Gao and Liu 2017]. However, our implementation is a RGB-D visual odometry system rather than a stereo visual odometry system. The visual odometry system is divided into parallel Frontend and Backend. The Visual Frontend is responsible for detecting and tracking feature points while localizing the camera pose in real-time. When the number of tracked feature points falls below a threshold, the Frontend inserts a new key frame, detects new features and triggers Backend optimization. The Backend jointly optimizes the pose of several local key frames and the positions of observed features by minimizing the coordinate difference.

1.1.1 Frontend. We used GFTT [Jianbo Shi and Tomasi 1994] for feature detection and Lucas-Kanade optical flow to track those features [Lucas et al. 1981]. We first initialized a map by projecting

Authors' address: Jianwei Zhang*; Fanwei Kong*; Chengjie Zhu, UC Berkeley.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
0730-0301/2020/8-ART111 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the detected feature points into 3D space using depth provided by the depth camera. We then tracked camera pose by minimizing the projection errors between the projected 3D map points and the associated image feature points.

1.1.2 Backend. We maintained a fixed window of 7 key frames in the Backend Optimization and applied Bundle Adjustment to those key frames and the associated landmarks. We adjusted camera poses and landmark positions to minimize the coordinate difference of the feature points in those frames. The Backend Optimization runs in parallel on a separate thread.

1.1.3 Optimization. We applied optimization to find optimal camera pose and landmark positions. The camera pose can be represented by a Special Euclidean group which comprises a rotation matrix R and translation vector t. The associated Lie Algebra can be represented by(ω, t) where $\omega, t \in R^3$. Rotation matrix R and Lie algebra representation ω follows exponential map and log map.

$$e^{\hat{\omega}} = R \tag{1}$$

$$\omega = \widehat{logR} \tag{2}$$

$\hat{\omega}$ is the skew symmetric matrix of vector ω.

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{3}$$

The error is defined as the difference of 3D coordinates in the camera space.

$$error = RX_{map} + t - X_{observed} \tag{4}$$

The Jacobian matrix is

$$\frac{\partial error}{\partial(\delta\omega, \delta t)} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & 1 & 0 & 0 \\ \omega_3 & 0 & -\omega_1 & 0 & 1 & 0 \\ -\omega_2 & \omega_1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where $(\delta\omega, \delta t)$ is the left-multiply update on the original $se3$.

$$\omega_{updated} = \widehat{e^{\delta\omega} \cdot e^{\hat{\omega}}} \tag{6}$$

$$t_{updated} = e^{\widehat{\delta\omega}} \cdot st + \delta t \tag{7}$$

We can use the similar principle to derive the Jacobian Matrix to calculate 2D projection error and map point positions. We used Levenberg–Marquardt algorithm to minimize the squared error and applied Huber robust kernel to guard against outliers. Observations with large loss were rejected during optimization. The optimization framework was implemented in g2o [Kümmerle et al. 2011].

1.2 Mesh Reconstruction

1.2.1 TSDF. We applied the TSDF algorithm to reconstruct a 3D voxel grid from multiple input frames. With the estimated camera poses, we can compute the world coordinates of each pixel in the RGB-D images and thus align RGB-D images from different frames into a consistent space. Let $\mathbf{u}$ denote the image coordinates, $D_k(\mathbf{u})$ denote the per-frame depth image from frame k of capture and K denote the camera intrinsic matrix, the camera-relative coordinates are

$$\mathbf{v}_k(\mathbf{u}) = D_k(\mathbf{u}) \odot \left(K^{-1}\dot{\mathbf{u}}\right) \tag{8}$$

where $\dot{\mathbf{u}} = \begin{bmatrix}\mathbf{u}^T & 1\end{bmatrix}^T$ is the image coordinates in homogeneous form. Let T_k denote the estimated camera pose (camera-to-world transformation) at frame k, we can compute observed points on images in global coordinate frame as,

$$\mathbf{v}_k^g(\mathbf{u}) = T_k\dot{\mathbf{v}}_k(\mathbf{u}) \tag{9}$$

where $\dot{\mathbf{v}}_k(\mathbf{u}) = \begin{bmatrix}\mathbf{v}_k(\mathbf{u})^T & 1\end{bmatrix}^T$ is the camera-relative coordinates in the homogeneous form.

We then calculated the sign distance value of each voxel in the 3D voxel grid. Let $\mathbf{p}$ denote the projected grid point of pixel $\mathbf{u}$ in the global coordinate space, the distance value d at this grid point is defined as,

$$d = D_k(\mathbf{u}) - d_p \tag{10}$$

where d_p is the distance between this point and the camera. We truncated the distance value to reduce the number of grids we need to march later to reconstruct the mesh. Namely, the distance value was truncated by a threshold μ,

$$d_t = \max(\min(1, \frac{d}{\mu}), -1) \tag{11}$$

At each frame, we calculated the distance values at the projected grid points of all the pixels captured and fuse the distance values from different frames by computing their running averages $F_k(\mathbf{p})$,

$$F_k(\mathbf{p}) \coloneqq \frac{W_{k-1}(\mathbf{p})F_{k-1}(\mathbf{p}) + W_k(\mathbf{p})F_k(\mathbf{p})}{W_{k-1}(\mathbf{p}) + W_k(\mathbf{p})} \tag{12}$$

$$W_k(\mathbf{p}) \coloneqq W_{k-1}(\mathbf{p}) + W_k(\mathbf{p}) \tag{13}$$

where $W_k(\mathbf{p})$ is the weight assigned to the grid point $\mathbf{p}$ at frame k. The weight is computed as,

$$W_k(\mathbf{p}) = \lambda\frac{\cos(\theta)}{D_k(\mathbf{u})} \tag{14}$$

which weights points that are closer and away from camera glancing angles more heavily.

Since a large portion of the 3D TSDF voxel grid does not contain objects in the scene, it is inefficient to represent the scene using a dense TSDF voxel grid. A sparse representation of the TSDF volume can improve the speed and memory efficiency of TSDF fusion. We integrated the scalable implementation of TSDF fusion in Open3D into our framework [Zhou and Koltun 2013; Zhou et al. 2018]. This implementation constructed a sparse representation of TSDF volume by using unordered map to build a hierarchical hashing table that only stores voxels with distance values d_t between 1. and -1.

1.2.2 Marching Cube. We applied the marching cube algorithm to extract surfaces from the fused TSDF volume [Lorensen and Cline 1987]. Since we have divided the space into a 3D voxel grid when calculating the TSDF volume, we applied marching cube on the same voxel grid. Briefly, the algorithm iterates over each voxel stored in the unordered map and checks the value at each voxel corner against the iso-value to determine whether the corner is inside the geometry. We used an iso-value of 0. in our framework. The 8 scalar values are then encoded to identify the polygon configuration at this voxel.

2 RESULTS

2.1 Visual Odometry

Our simple visual odometry framework was able to successfully detect feature points and estimate camera poses. Figure 2 shows the detected feature points on the images and the estimated camera poses from the feature points. Although our framework does not require reconstructing point clouds to extract surface mesh, we plotted the point clouds aligned with the estimated camera poses to visualize the effectiveness of our visual odometry framework. The points were colored based on their corresponded pixel colors.

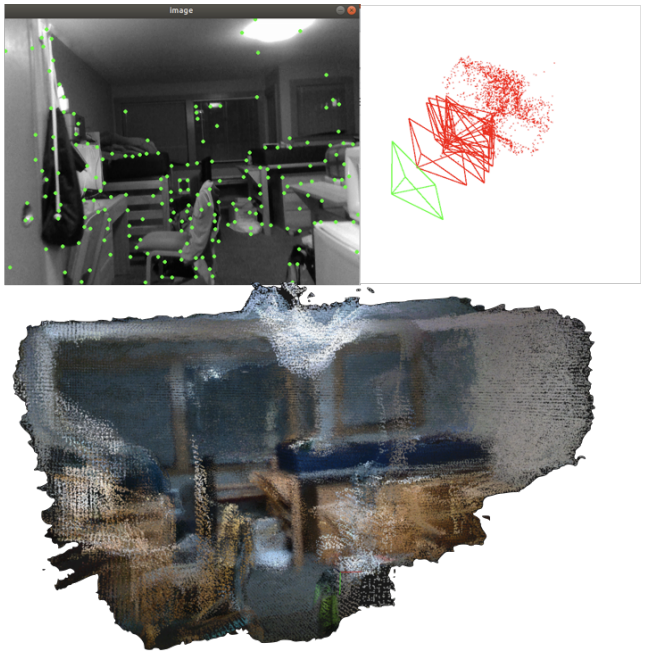

Fig. 2. Visual odometry using bundle adjustment: feature point detection (left), estimated camera pose (right), and aligned point cloud using estimated camera poses (bottom).

2.2 Mesh Reconstruction

2.2.1 Final Results. We obtained RGB-D images of two scenes, a dorm room and a corridor. Figure 3 displays our reconstruction results of the dorm room and the corridor and provided zoomed-in views of the detailed features in the scenes. We used a voxel length of 0.007m and a truncation threshold of 0.11m to generate the meshes. With these settings, we were able to clearly visualize the the small objects in the reconstructed scenes, such as cables, the computer mouse, cups and poster on the door.

2.2.2 Experiments. While we worked towards reconstructing high quality meshes of the scenes, we identified and experimented with some factors that could affect mesh reconstruction results.

Camera motion blur can lead to inaccurate pose estimation and mesh reconstruction. We explored different camera exposure and gain settings to reduce the motion blur in the images captured. As shown in figure 4, using a shorter exposure time of 0.15ms and a larger gain of 128 produced sharper images with less motion blur, compared with using an exposure time of 0.5ms and gain of 4. We also compared the reconstructed meshes using images taken with different camera settings. As labeled in the bottom panel of figure 4, we observed differences in reconstructed meshes, especially for the mattress corner and the microwave. With less motion blur, the reconstructed object surfaces seemed to have sharper edges. We note that the meshes shown in this figure were reconstructed using only 16 frames and thus seemed to be noisy for both camera settings. We expect more notable differences when more frames are fused into the TSDF volumes, which can alleviate the noise.

Fig. 3. Reconstructed mesh of a dorm room (top) and a corridor (bottom). Zoomed-in views displayed on the bottom right corners.

The quality of mesh reconstruction depended on the voxel length of the TSDF grid and the truncation threshold we used to calculated the sign distance value d_t. Figure 5 displays the effects of different combination of TSDF voxel length and truncation threshold on our reconstruction results. With a smaller voxel length of the TSDF grid, we were able to reconstruct the scene at higher resolution and capture the detailed features of some small objects in the scene. A smaller truncation threshold produced a similar effect. This is expected since with a larger truncation threshold, more voxels will be included into the running average calculation of the TSDF volume. As shown in the upper right panel of 5, with a large voxel length and a small truncation threshold, the reconstructed mesh had a lot of holes. Therefore, to reconstruct the scene at a lower resolution, it is important to increase the truncation threshold accordingly.

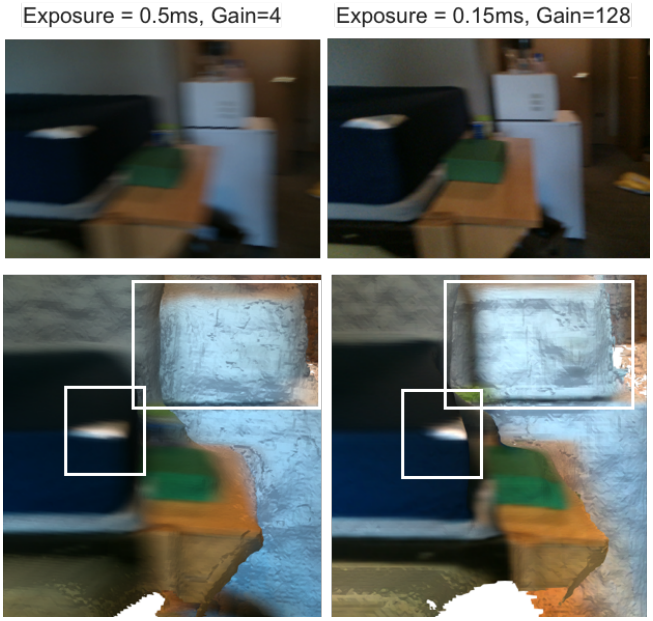

Fig. 4. Captured images (top) and reconstructed scenes (bottom) using different camera exposure settings.

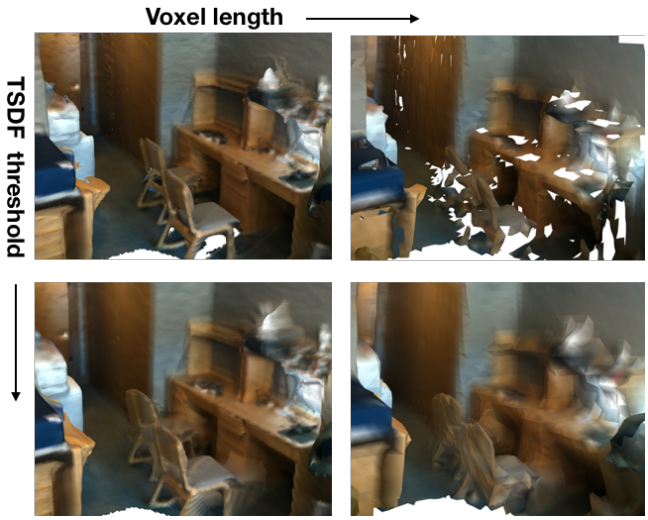

Fig. 5. Effects of TSDF truncation threshold and voxel length mesh reconstruction results.

We explored the effect of using different TSDF parameters on computing time of TSDF fusion and mesh generation using marching cube. We also compared the time spent in mesh reconstruction and visual odometry, respectively. All timings were performed on a 3GHz Intel Core i5 CPU. As shown in figure 6, using a larger truncation threshold or smaller voxel length increased the computing time of both TSDF fusion and marching cube. The increase in computing time with respect to voxel length was more drastic, and that increase in time was mostly due to surface generation using the marching cube algorithm. Overall, the marching cube was the most time consuming step in our framework, followed by TSDF fusion and then visual odometry. Especially for larger scenes, it became the major bottleneck preventing our framework from being executed in real-time. Therefore, we decided to provide the option of only extracting the surfaces at the last frame or after a certain number of frames. With a high resolution TSDF grid (with a voxel length of 0.007m), our framework can execute visual odometry and TSDF integration at approximately 3 fps.

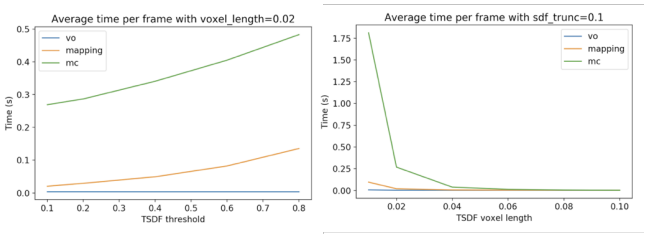

Fig. 6. Effect of TSDF truncation threshold (left) and voxel length (right) on computational time of visual odometry (vo), TSDF fusion (mapping) and marching cube (mc).

The weighted average of TSDF provides the advantage of smoothing out noises and producing a smooth surface output, even for noisy depth images. It can be robust to outlier structures like floating noise pixels and bumps along structure edges. We visualized this nice property of TSDF fusion in figure 7, where the reconstructed meshes from the first and the last frame were displayed. After fusing many frames into the TSDF volume, the marching cube algorithm produced meshes with higher fidelity.

3 CONCLUSION AND FUTURE WORK

We have successfully built a 3D scene reconstruction system from RGB-D images by applying visual odometry and mesh reconstruction techniques. In contrast to existing complex 3D scene reconstruction systems, our system is relatively simple and built on concepts that we can easily understand. Our visual odometry framework can reliably and efficiently estimate camera pose from multiple frames. The visual odometry system consists of parallel Frontend and Backend that can detect/track feature points and solve for optimized camera poses. The mesh reconstruction framework can generate surface mesh of the scene with detailed features by fusing the RGB-D images into a TSDF volume and applying the marching cube algorithm.

3.1 Reintegration

To achieve more accurate local mapping within our visual odometry framework, the Backend will need to jointly optimize the camera pose of several previous key frames. With a adjusted pose of previous frames, a TSDF reintegration may be necessary to produce more consistent results. Since this functionality requires large computation ability and should be carried out on GPU, we have left this part for future work.

Fig. 7. Comparison of reconstruction results at the first frame and the last frame.

3.2 Loop Closure

The system does not handle Loop Closure. Hence, it will fail when the camera performs large loopy motion. For small scale environment, such as a room, loop closure can be avoided by choosing proper camera trajectory. In the future, we can also add loop closure detection and apply global nonlinear optimization to align revisited locations [Zhou and Koltun 2013].

3.3 Realtime Performance

Our system runs on CPU and is accelerated by OpenMP. The visual odometry and TSDF integration parts of the framework are relatively efficient. It can currently run at approximately 10fps when doing incremental TSDF integration with a reasonable choice of voxel length and truncation threshold. Our system does not have GPU support due to time and hardware constrains of this project. TSDF integration can be easily parallelized on GPU and could improve our real-time performance. Surface extraction using the marching cube algorithm could also be parallelized in the future to be able to reconstruct surfaces in real time.

4 CODE

Our code can be found here.

REFERENCES

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Niessner. 2017a. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. 2017b. BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration. *ACM Trans. Graph.* 36, 3, Article 24 (May 2017), 18 pages. https://doi.org/10.1145/3054739

Jianbo Shi and Tomasi. 1994. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.* 593–600.

R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. 2011. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation.* 3607–3613.

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm.. In *SIGGRAPH*, Maureen C. Stone (Ed.). ACM, 163–169. http://dblp.uni-trier.de/db/conf/siggraph/siggraph1987.html#LorensenC87

Bruce D Lucas, Takeo Kanade, et al. 1981. An iterative image registration technique with an application to stereo vision. (1981).

Qinrui Yan Xiang Gao, Tao Zhang and Yi Liu. 2017. *14 Lectures on Visual SLAM: From Theory to Practice.* Publishing House of Electronics Industry.

Qian-Yi Zhou and Vladlen Koltun. 2013. Dense scene reconstruction with points of interest. *ACM Trans. Graph.* 32 (2013), 112:1–112:8.

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *ArXiv* abs/1801.09847 (2018).