UNIVERSITY COLLEGE LONDON, GEOMETRY PROCESSING

Report of the Kinect Fusion

Chengkun Li (ucabcl9@ucl.ac.uk)

April 30, 2021

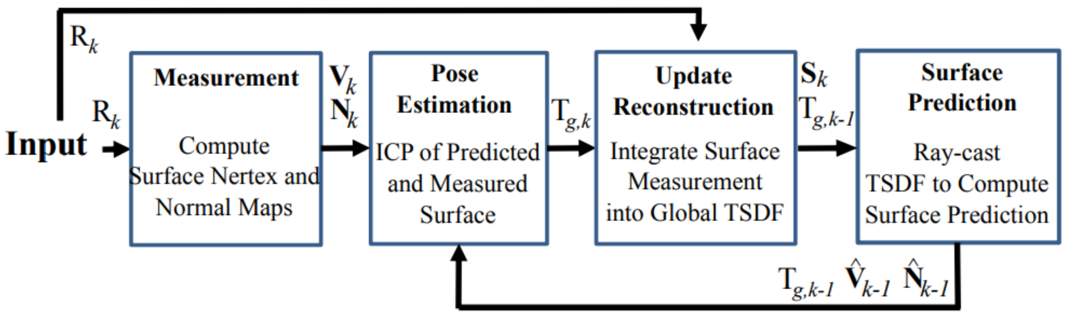
Abstract

Kinect Fusion is a method to rebuild 3D model of the scene by its depth image and RGB image which is collected by a RGB-D camera. The most famous approach is proposed by the paper "KinectFusion: Real-Time Dense Surface Mapping and Tracking" given by Izadi et al[1]. In this report, I implement the Kinect Fusion and propose some extensions to improve the results.

And I do this project with my teammate Ziian Guo (ucabzg4@ucl.ac.uk). However, we have a different understanding of Izadi's paper[1]. Actually, we respectively implement two different versions of Kinect Fusion and more details can be found in his report.

1. Paper Summary

The key contributions of this paper are (1) proposing a way to always update surface representation by fusing all the previous frame by the truncated signed distance function (TSDF); (2) tracking the camera pose by frame to model method; (3) using GPU to accelerate the algorithm. The overview of the method can be seen in Fig.1.



R_k
R_k
Input
Measurement
Compute Surface Nertex and Normal Maps
V_k
N_k
Pose Estimation
ICP of Predicted and Measured Surface
$T_{g,k}$
Update Reconstruction
Integrate Surface Measurement into Global TSDF
S_k
$T_{g,k-1}$
Surface Prediction
Ray-cast TSDF to Compute Surface Prediction
$T_{g,k-1}$ $\hat{V}_{k-1}$ $\hat{N}_{k-1}$

Fig.1. Overall system workflow

Surface Measurement: Using camera intrinsic matrix to compute vertex map ($\boldsymbol{V}_k$) and normal map (N_k) which are in camera space from the raw depth image *(R_k)* got by a Kinect device.

Camera Pose Estimation: Computing the camera pose matrix by iterative closest point (ICP) algorithm between frame k-1 and frame k. And then using $\boldsymbol{T}_{g,k} = \boldsymbol{T}_{g,k-1}\boldsymbol{T}_{k-1,k}$ to obtain the global transformation matrix.

Surface Reconstruction Updating: Integrating the measurement of frame k into the whole scene which is represented by a volume in which each voxel's value is obtained by computing TSDF.

Surface Prediction: The final result is shown by ray-casting TSDF to compute the surface in the whole scene (the volume). At the same time, we can generate a close loop, by getting a prediction vertex map ($\hat{\boldsymbol{V}}_k$) and normal map ($\hat{\boldsymbol{N}}_k$) of frame k from ray-casting as the previous knowledge for the next frame's pose estimation. Actually, this is a frame to model method, because $\hat{V}_k$, $\hat{N}_k$ is computed by our fused scene.

2. The Implementation of Kinect Fusion

2.1 The Dataset

For testing our algorithm, we use the TUM dataset[1], which contains the color and depth images of a Microsoft Kinect sensor along the ground-truth trajectory of the sensor. The data was recorded at full frame rate (30 Hz) and sensor resolution (640x480). And there is an example shown in Fig.1.

[1] https://vision.in.tum.de/data/datasets/rgbd-dataset/download

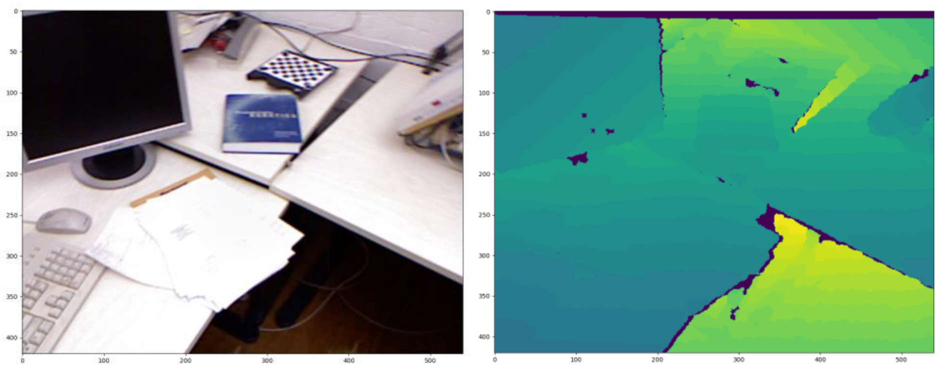


Fig.1. The TUM dataset. Left: the color image; Right: the depth image.

2.2 Surface Measurement

In this paper, author use a bilateral filter to obtain a denoised depth map $\mathrm{D_k}$ from the raw depth image $\mathrm{R_k}$. We can write as,

$$D_k(\boldsymbol{u}) = f_{denoising}(R_k(\boldsymbol{u}))$$

where $f_{denosing}(\cdot)$ represents bilateral filtering operation. Image pixel $\boldsymbol{u} = (u,v)^T$ is in the image domain $\boldsymbol{u} \in \boldsymbol{R}^2$.

Then we back-project the filtered depth values into the sensor's frame by using the camera's intrinsic matrix **K**.

$$\boldsymbol{V}_k(\boldsymbol{u}) = D_k(\boldsymbol{u})\boldsymbol{K}^{-1}\dot{\boldsymbol{u}}$$

where $\dot{\boldsymbol{u}}$ is the homogeneous vector of $\boldsymbol{u}$ ($\dot{\boldsymbol{u}} \coloneqq \left(\boldsymbol{u}^{\mathrm{T}}|1\right)^{\mathrm{T}}$). Easily, we can also get the normal vector of each $V_k(\boldsymbol{u})$ point by

$$\boldsymbol{N}_k(\boldsymbol{u}) = v[(\boldsymbol{V}_k(u+1,v) - \boldsymbol{V}_k(u,v) \times (\boldsymbol{V}_k(u,v+1) - \boldsymbol{V}_k(u,v)))]$$

where $v(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\|x\|_2}$ is a normalizing function.

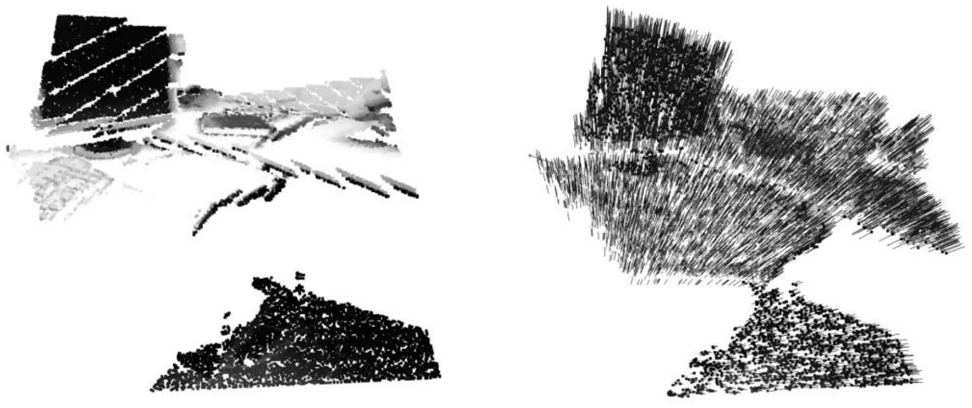


Fig.2 The point cloud in the sensor's frame. Left: the vertex map; Right: the normal map.

An example of the surface measurement is shown in Fig.2. Actually, because of the resolution of the Kinect sensor, the depth image is not continuous and it is more like striped (like the depth image in Fig.1). So, the point cloud we get is also striped, and the normal vectors are orientated to the same direction.

2.3 Camera Pose Estimation

Considering the high frame rate (30 Hz), we can assume small motion from one frame to the next. This allow us to use ICP algorithm to compute the transformation matrix $\boldsymbol{T}_{k-1,k} \in \boldsymbol{R}^{4\times4}$ between two adjacent frames

$$\boldsymbol{T}_{k-1,k} = ICP(\boldsymbol{V}_{k-1}, \boldsymbol{N}_{k-1}, \boldsymbol{V}_k, \boldsymbol{N}_k)$$

And each global frame surface prediction can be obtained using the previous fixed pose estimate $\boldsymbol{T}_{g,k-1}$.

$$\boldsymbol{T}_{g,k} = \boldsymbol{T}_{k-1,k}\boldsymbol{T}_{g,k-1}$$

Especially, we set $\boldsymbol{T}_{g,1} = \boldsymbol{I}$ which $\boldsymbol{I}$ is an identity matrix for the first frame.

This method is called frame-to-frame tracking. However, if we use $(\widehat{\boldsymbol{V}}_{k-1}, \widehat{\boldsymbol{N}}_{k-1})$ to replace $(\boldsymbol{V}_{k-1}, \boldsymbol{N}_{k-1})$, we note this method as frame-to-model tracking, where $(\widehat{\boldsymbol{V}}_{k-1}, \widehat{\boldsymbol{N}}_{k-1})$ is the result of fusion (more details are shown in section 2.5). Thus the ICP function converts into:

$$\boldsymbol{T}_{k-1,k} = ICP(\widehat{\boldsymbol{V}}_{k-1}, \widehat{\boldsymbol{N}}_{k-1}, \boldsymbol{V}_k, \boldsymbol{N}_k)$$

One thing worth to mention is that the point-to-plane ICP is proposed to be used. However, I found $\mathbf{N}_\mathrm{k}$ almost orientates to the same direction which could be seen in Fig.2. Thus, the point-to-plane ICP is useless in this case, and the point-to-point version will reach the same result.

2.4 Surface Reconstruction Updating

In Kinect Fusion, each consecutive depth frame, with an associated live camera pose estimate, is fused incrementally into one single 3D reconstruction using the volumetric truncated signed distance function (TSDF).

Assuming the true depth value lies within ±μ of the measured value $\mathrm{R}_\mathrm{k}(\boldsymbol{u})$, then for a distance r from the camera center along each depth map ray to the surface, we can get $|r - \lambda R_k(\boldsymbol{u})| \leq \mu$) (here $\lambda = ||\boldsymbol{K}^{-1}\dot{\boldsymbol{u}}||$ scales the measurement along the pixel ray). The TSDF field is obtained by truncating the SDF filed in which the value of each point represents the distance to the nearest surface point. And, for points within visible space at distance greater than μ from the nearest surface interface are truncated to a maximum distance μ. Non-visible points farther than μ from the surface are not measured. Fig.3 shows a visual example of TSDF.

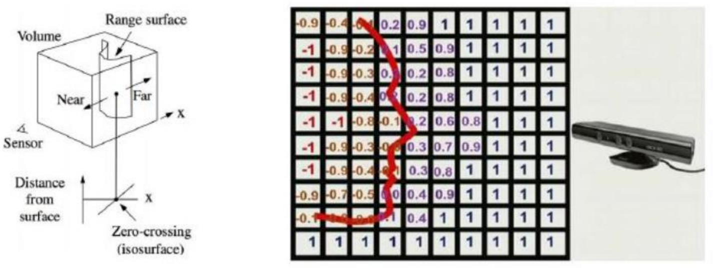


Fig.3 The truncated signed distance function

For a raw depth map $\mathrm{R_k}$, the TSDF value $F_{R_k}(\boldsymbol{p})$ of each point $\boldsymbol{p} = (x, y, z)^T$ in the global frame can be computed by the following functions.

$$F_{R_k}(\boldsymbol{p}) = \Psi\left(\lambda^{-1}\left\|\boldsymbol{t}_{g,k} - \boldsymbol{p}\right\|_2 - R_k(\boldsymbol{x})\right)$$

$$\lambda = \|\boldsymbol{K}^{-1}\dot{\boldsymbol{x}}\|_2$$

$$\boldsymbol{x} = \left\lfloor \pi\left(\boldsymbol{K}\boldsymbol{T}_{g,k}^{-1}\boldsymbol{p}\right)\right\rfloor$$

$$\Psi(\eta) = \begin{cases} \min\left(1, \frac{\eta}{\mu}\right) sgn(\eta) & iff\ \eta \geq -\mu \\ null & otherwise \end{cases}$$

where $\pi(\cdot)$ function is a dehomogeneous operation which performs perspective projection of $(\mathrm{x,y,z})^{\mathrm{T}}$ and gets $\left(\frac{\mathrm{x}}{\mathrm{z}}, \frac{\mathrm{y}}{\mathrm{z}}\right)^{\mathrm{T}}$, and $\lfloor\cdot\rfloor$ is the floor function which is used to prevent smearing of measurements at depth discontinuities. So, $\boldsymbol{x}$ represents the u-v coordinates to which the point $\boldsymbol{p}$ can project in $\mathrm{R_k}$ frame. $\Psi(\cdot)$ is used to truncate SDF to TSDF. $\left\|\boldsymbol{t}_{g,k} - \boldsymbol{p}\right\|_2$ represents the distance between the point $\boldsymbol{p}$ and the camera centroid, and $\frac{1}{\lambda}$ converts the ray distance to a depth value. And you can get $\boldsymbol{t}_{g,k}$ from the sensor pose $\boldsymbol{T}_{g,k}$, where

$$\boldsymbol{T}_{g,k} = \begin{pmatrix} \boldsymbol{R}_{g,k} & \boldsymbol{t}_{g,k} \\ \boldsymbol{0} & 1 \end{pmatrix} \in \mathrm{SE3}$$

The fusion processing can be write as:

$$F_k(\boldsymbol{p}) = \frac{W_{k-1}(\boldsymbol{p})F_{k-1}(\boldsymbol{p}) + W_{R_k}(\boldsymbol{p})F_{R_k}(\boldsymbol{p})}{W_{k-1}(\boldsymbol{p}) + W_{R_k}(\boldsymbol{p})}$$

$$W_k(\boldsymbol{p}) = W_{k-1}(\boldsymbol{p}) + W_{R_k}(\boldsymbol{p})$$

The associated weight $W_k(\boldsymbol{p})$ represents the fusion weight of the point $\boldsymbol{p}$. And the fusion processing only occurs when $F_{R_k}(\boldsymbol{p})$ is not null and less than 1. And finally, we will get a TSDF volume F_k shown in Fig.4.

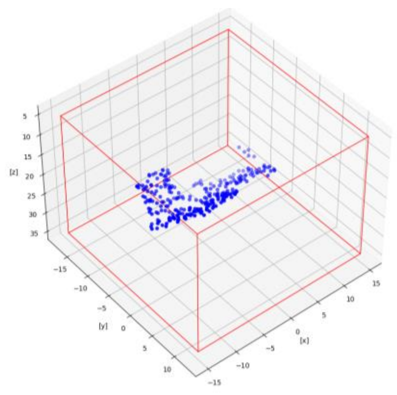


Fig.4. The volume space to compute TSDF

2.5 Surface Prediction

In this part, using ray-casting to reconstruct a dense surface from the TSDF field F_k by find the iso-surface where $F_k(\boldsymbol{p}) = 0$. The pixel-ray, $\mathbf{T}_{g,\boldsymbol{k}}\boldsymbol{K}^{-\mathbf{1}}\dot{\boldsymbol{u}}$, is starting from the sensor's centroid and stopping at the object's surface, which makes a zero-crossing (from $+ve$ to $-ve$, or from $-ve$ to $+ve$). By finding these zero-crossing point's position, we can obtain the vertex map $\widehat{\boldsymbol{V}}_k$. Easily, for $\widehat{\boldsymbol{N}}_k$, we compute:

$$\widehat{\boldsymbol{N}}_k(\boldsymbol{u}) = \boldsymbol{R}_{g,k}^{-1}\widehat{\boldsymbol{N}}_k^g(\boldsymbol{u})$$
$$\widehat{\boldsymbol{N}}_k^g(\boldsymbol{u}) = v\big(\nabla F(\boldsymbol{p})\big)$$
$$\nabla F = \left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right]^T$$

where $v(\cdot)$ is the normalizing function, $\boldsymbol{R}_{g,\boldsymbol{k}}$ is the rotation transformation matrix which can be obtained by $\boldsymbol{R}_{g,k}$.

However, in practice, we can obtain the iso-surface by marching cubes algorithm, and more details can be found in Wikipedia[2]. Fig.5 shows a fusion result which fuses 10 frames using a reconstruction resolution of 256^3.

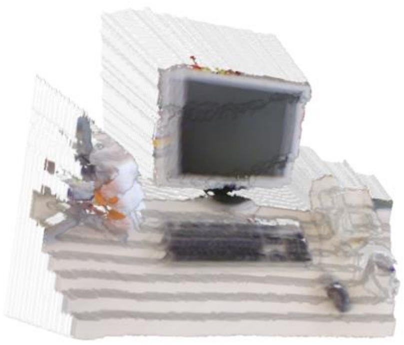


Fig.5. The result of fusion by 10 frames

[2] https://en.wikipedia.org/wiki/Marching_cubes

3. Evaluation

I have conducted a number of experiments. However, I did not implement it by GPU, because of the low efficiency for python loop. I just test it with few frames and low resolution. (Actually, you can use GPU and numba model to accelerate).

Fig.6 shows us the fusion performance with different number of frames. With the increasing of frames, the rendering scene is extending and the reconstructed surface is fused. However, some error will be also imported because the ICP registration is not so perfect.

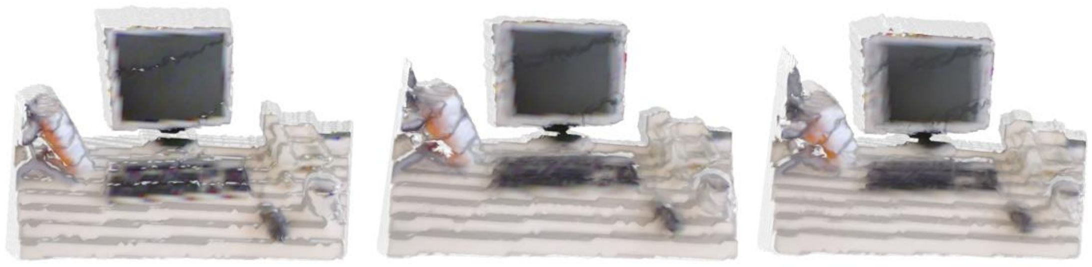


Fig.6. Fusing with different number of frames. Left: 2 frames; Medium: 5 frames; Right: 10 frames. (The volume resolution is 128^3 with truncation distance $\mu = 1$)

Then, I test Kinect Fusion with different volume resolution. And with the higher resolution, more details will be shown in the fusion result.

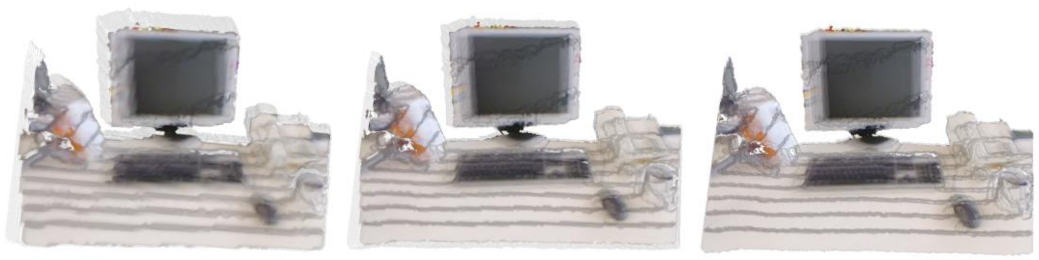


Fig.7. Fusion using different volume resolution. Left: 128^3; Medium: 256^3; Right: 512^3.

(Fusing with 10 frames)

Finally, I try to find the influence of truncation distance. If the truncation distance is too small, it will make the fusion result fragmental.

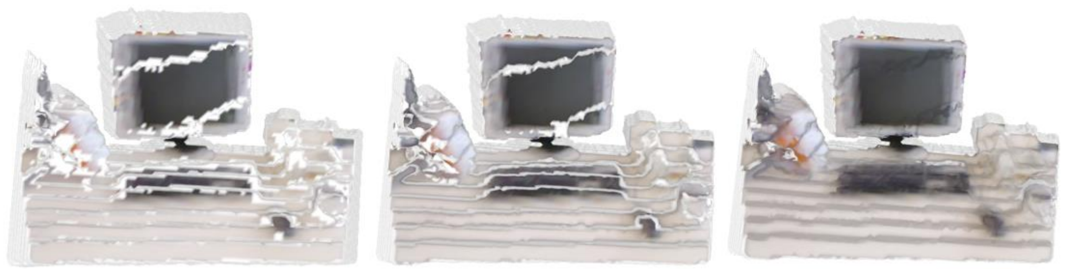


Fig.8. Fusion with different truncation distance μ. Left: $\mu = 0.1$; Medium: $\mu = 0.5$; Right: $\mu = 1$. (The volume resolution is 128^3 and fuse with 10 frames)

4. Paper Criticism

There are several disadvantages in this algorithm.

Firstly, we assume that there is only a small movement between the adjacent frames. Although, this is correct in the majority of situations because of the high frame-rate sensor. There is also probability that the sensor's pose changes too fast so that ICP registration fails.

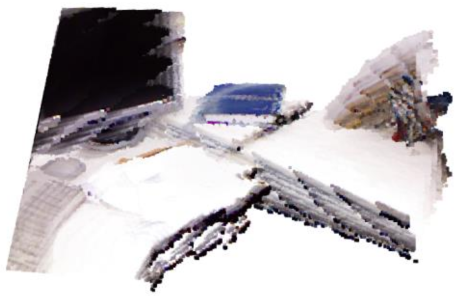


Fig.9. ICP registration failed

Secondly, in the original paper, they did not render the reconstructed surface with color. Actually, we can use the color image information to do color fusion for the better performance.

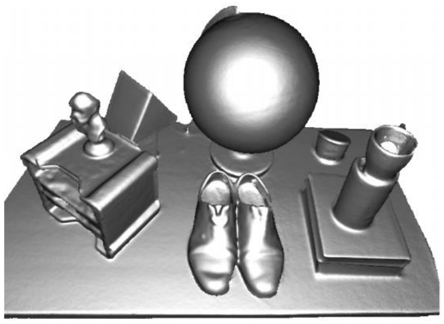


Fig.10. No rendering color[3]

Finally, the performance is associated with the resolution of the TSDF volume space you defined (Fig.4). However, time complexity and space complexity increase with the increasing resolution. Apart from this, you must define the scale of volume firstly, and the part that is out of the volume's bound can not be tracked.

[3] This picture is from "KinectFusion: Real-time dense surface mapping and tracking"

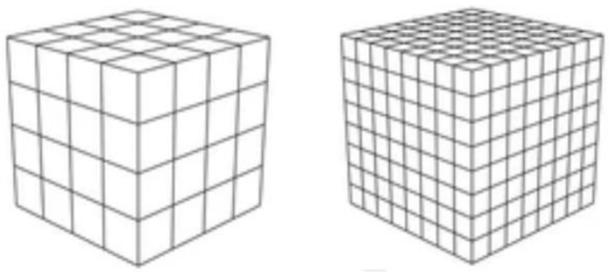


Fig.11. Resolution increasing

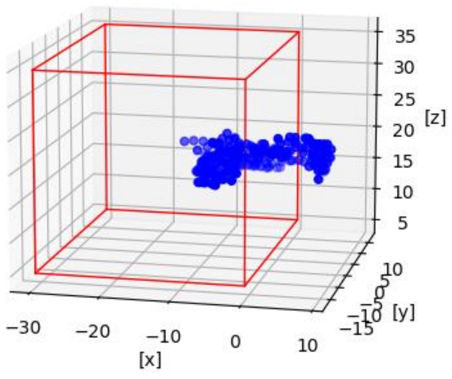


Fig.12. Out of bound

5. My Extension

For solving the problems I proposed, I have done several improvements.

5.1 Colored Point Cloud Registration

We can estimate the sensor's pose using two colored point clouds by Park's method [2]. And this algorithm has been implemented by Open3d. It is clearly that Park's method shows better performance and more robust.

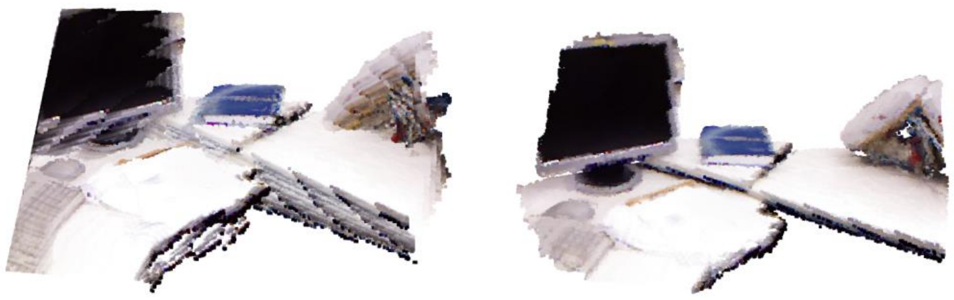


Fig.13. Registration. Left: ICP method; Right: Park's method.

5.2 Colorful Fusion

In the original KinectFusion paper, the author did not implement the color rendering. So, for this part, I implement two ways to color the reconstructed surface. I call these two methods as frame method and volume method respectively.

5.2.1 The frame method

One way is to compute the color of each fused point by frames. After fusion, we can get fused surface vertex $\hat{\boldsymbol{v}}$. And then match this vertex into color frames to obtain this point's color $C_k(\hat{\boldsymbol{v}})$ of frame k. And we can do following to compute color.

$$\boldsymbol{x} = \lfloor \pi(\boldsymbol{K}\boldsymbol{T}_{g,k}^{-1}\hat{\boldsymbol{v}}) \rfloor$$
$$C_k(\hat{\boldsymbol{v}}) = RGB_k(\boldsymbol{x})$$

where RGB_k is the color image of frame k. Finally, we compute the average color.

$$C(\hat{\boldsymbol{p}}) = \frac{1}{\mathrm{n}}\sum C_k(\hat{\boldsymbol{p}})$$

Fig.14 shows the colored result by frame method (here we set volume resolution as 128^3 with truncation distance $\mu = 1$)

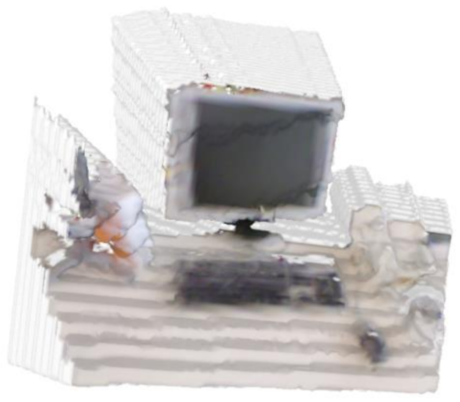


Fig.14. Colored by frame method

5.2.2 The volume method

Another way to do it is to estimate color by the volume. When computing the TSDF value of the volume's voxel, we estimate the voxel's color $C_{R_k}(\boldsymbol{p})$ at the same time. This processing can be described by following functions.

$$C_{R_k}(\boldsymbol{p}) = \Psi\left(\lambda^{-1}\left\|\boldsymbol{t}_{g,k} - \boldsymbol{p}\right\|_2 - R_k(\boldsymbol{x})\right) \times RGB_k(\boldsymbol{x})$$
$$\lambda = \|\boldsymbol{K}^{-1}\dot{\boldsymbol{x}}\|_2$$
$$\boldsymbol{x} = \lfloor \pi(\boldsymbol{K}\boldsymbol{T}_{g,k}^{-1}\boldsymbol{p}) \rfloor$$
$$\Psi(\eta) = \begin{cases} 1 & if\ -\mu \le \eta \le +\mu \\ null & otherwise \end{cases}$$

And the update processing is the same with TSDF.

$$C_k(\boldsymbol{p}) = \frac{W_{k-1}(\boldsymbol{p})C_{k-1}(\boldsymbol{p}) + W_{R_k}(\boldsymbol{p})C_{R_k}(\boldsymbol{p})}{W_{k-1}(\boldsymbol{p}) + W_{R_k}(\boldsymbol{p})}$$

Where $W_k(\boldsymbol{p})$ is the same with TSDF processing.

For do marching cubes, we need to solve interpolation problem, how to estimate the color for the point that is not the vertex of the volume. Here, we use inverse distance weighted method to compute color. For an arbitrary point, we can find 8 nearest vertices like Fig.6. So the color of this point $C_k(\boldsymbol{v})$ is

$$C_k(\boldsymbol{v}) = \frac{\sum_{n=1}^{8} \frac{1}{d_n} \times C_k(\boldsymbol{p}_n)}{\sum_{n=1}^{8} \frac{1}{d_n}}$$

where d_n is the distance between point $\boldsymbol{v}$ and vertex $\boldsymbol{p}_n$, the dotted line in Fig.6.

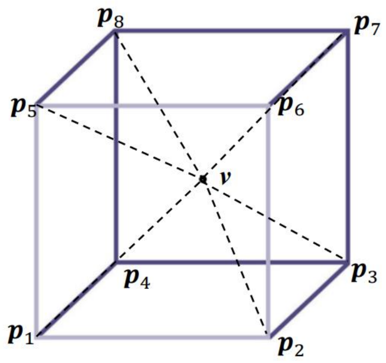


Fig.15. Color interpolation problem

Fig.15 shows the colored result by volume method (here we set volume resolution as 128^3 with truncation distance $\mu = 1$)

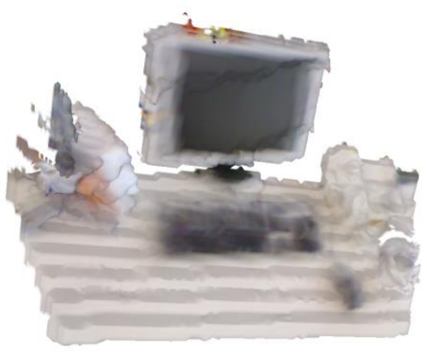


Fig.16. Colored by volume method

In conclusion, there is almost no difference between our frame method and volume method for the rendering performance. However, the time and space complexity of volume method is less than the frame method. Thus, I propose more to use my volume method.

5.3 Fusion With The Point Cloud

Unlikely TSDF method, we don't do ray casting for the TSDF field to find the fused surface. We can fusion the point cloud's position directly. But, we need to justify which points should be fused firstly. And if some points of the previous frame's fused point cloud could project on the current frame's u-v coordinate, and the projected depth value and the raw depth value of current frame satisfies $|r_{\text{projected}} - r_{\text{raw}}| \leq \mu$, then we can fuse these points with their corresponding points. This processing can write as following functions:

$$Flag = \Psi\left(\lambda^{-1}\left\|\boldsymbol{t}_{g,k} - \boldsymbol{p}\right\|_2 - R_k(\boldsymbol{x})\right)$$
$$\lambda = \|\boldsymbol{K}^{-1}\dot{\boldsymbol{x}}\|_2$$
$$\boldsymbol{x} = \lfloor \pi(\boldsymbol{K}\boldsymbol{T}_{g,k}^{-1}\boldsymbol{p}) \rfloor$$
$$\Psi(\eta) = \begin{cases} True & iff\ -\mu \leq \eta \leq +\mu \\ False & otherwise \end{cases}$$

Where $\boldsymbol{p}$ is one point of the point cloud which has been fused by previous k-1 frames. And $\boldsymbol{x}$ is the projected u-v coordinate for point $\boldsymbol{p}$ on frame k. If $Flag$ is $True$, we do fuse as following.

$$\widehat{\boldsymbol{p}} = \frac{W \times p + V_{R_k}(\boldsymbol{x})}{W + 1}$$
$$\widehat{W} = W + 1$$

Where $V_{R_k}(\boldsymbol{x})$ is the corresponding vertex of $\boldsymbol{x}$, and the detail for this part has been described in section 2.2. For the u-v coordinates $\overline{\boldsymbol{x}}$ which don't have corresponding points in previous fused point cloud, we directly add them into the fused point cloud set. And final fused result is shown in Fig.17.

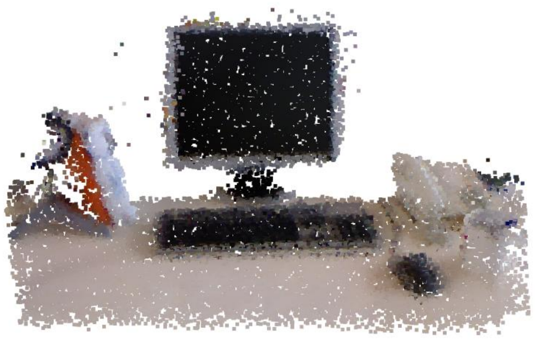


Fig.17. Fusion with the point cloud

Compare to the TSDF method the paper proposed, this method does not need to define a volume previously, and we do not need to confuse with the resolution of the volume. If the number of frames is not too big, the efficiency of this method is better than TSDF method, because we do fuse directly on the point cloud and do not need to compute the TSDF value for all of the vertices of volume. However, with the frame's number increasing, the time and space complexity will increase because of the point cloud's size larger and larger.

Reference

[1] R. A. Newcombe *et al.*, "KinectFusion: Real-time dense surface mapping and tracking," *2011 10th IEEE Int. Symp. Mix. Augment. Reality, ISMAR 2011*, pp. 127–136, 2011.

[2] J. Park, "Colored Point Cloud Registration Revisited."