3D Surface Reconstruction Based on Kinect Sensor

Song Tiangang, Lyu Zhou, Ding Xinyang, and Wan Yi

***Abstract*—Traditional accurate 3D reconstruction can be roughly divided into two categories according to types of sensor. One is through expensive laser scanner, and the other is by photogrammetry, which perform the reconstruction using photos taken from different angles. However, in recent years, the rapid development of depth camera has threw some new light upon the field. In this paper, we present a 3D reconstruction plan for indoor scenes using a very low-cost depth camera, the Microsoft Kinect sensor. Our system include four steps: data preprocessing, pose estimation of the sensor, fusion of the depth data, 3D surface extraction. Firstly, we project each frame of depth data back into the space to obtain the point-cloud. Secondly, estimate the sensor pose using ICP algorithm. Thirdly, fuse all of the depth data using a volumetric method, and finally extract 3D surface from the global model.**

***Index Terms*—3D reconstruction, 3D registration, depth camera, kinect sensor, volumetric representation.**

I. Introduction

In this paper, we begin with the introduction of Microsoft Kinect Sensor, then describe the 3D reconstruction route we adopted. Kinect Sensor reaches a depth measurement accuracy of 4 millimeters at a relatively wide range from 0.8 to 3.5 meters, and can generates the depth image at a rate of 30 frames per second. With all these characteristics and a market price lower than $200($250 for the new Windows version), Kinect Sensor suddenly become a consumer-level product, making it possible for almost everyone to afford and doing researches.

II. Background

A. Methods for 3D Reconstruction

According to the equipment used, traditional methods for acquiring dense point-cloud of object can be categorized into two main streams. i.e. the multi-view stereo route and the 3D laser scanner route. However, gradually developed in the recent ten years, depth camera has become one of the popular 3D measurement equipment with advantages of its small size and ability to perform real-time data acquisition.

1) *Multi-view stereo*

Stereo matching is the process of taking two or more images of the same object from different views with a certain degree of overlap, finding corresponding points within the overlapping region to estimate the relative position of the cameras and reconstructing the 3D coordinate of the object. Its principle is shown in Fig. 1. This method has been widely used in photogrammetry since the analytical photogrammetry became popular in the 1980s[1]. The basic workflow of stereo matching is of two steps. The first step is to estimate the 3D pose of the camera, which is also known as relative orientation. Following that, with certain control information, such as control points, we acquire 3D spatial coordinate of the measured point using space intersection.

As a matter of fact, many sorts of factors influence the final accuracy [2], which can be classified on the basis of processing procedure as follows: First, the quality of the image point coordinate, such as the performance of the camera and the accuracy of the calibration. Second, shooting condition, method and way of control, such as the length of the baseline and the distribution of the control points. Third, the performance of the image processing and photogrammetry processing software and hardware based.

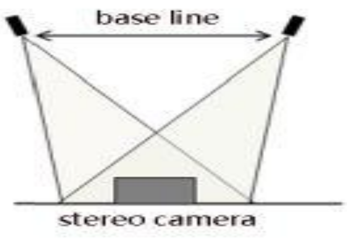

Fig. 1. Principles of multi-view stereo.

2) *Laser scanner*

Basic structures of a laser scanner includes laser light source that can beam laser, sensor that measures the distance to the surface, control unit and so on. Almost all types of laser scanner measure the distance by timing the round-trip time of a pulse of light, which is called the time of flight method as shown in Fig. 2. i.e., a laser is used to emit a pulse of light and the amount of time before the reflected light is seen by a detector is timed. Since light travels too fast, thus the requirement for measurement of the time must be extremely strict, which lead to the high price of laser scanner, ranging from $10,000 to $10,000,000. However, the accuracy of laser scanner is very attractive, ranging from millimeter level to micron or even sub-micron level. Limited by its price and principle, laser scanner usually serves in industrial measurement and large enterprises.

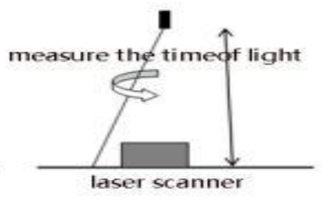

Fig. 2. Principles of laser scanner.

3) *Depth camera*

Depth camera can collect video the same as a general RGB camera, the only difference lies in the pixel of the image taken by the depth camera: rather than the color of objects, it takes down the distance (depth) between the object and the

Manuscript received October 9, 2012; revised December 18, 2012.
The authors are with the School of Remote Sensing and Information Engineering, Wuhan University, Wuhan, Hubei, China.

camera on the basis of TOF, as is shown in Fig. 3. The most influential depth camera is perhaps Mesa Imaging SwissRanger 4000(SR4000), which has a scan range of 5 to 8 meters, 176×144 pixel resolution and can operates up to 54fps. Different from laser scanner, the portability and real-time response of depth camera bring entirely new opportunity for the development of Simultaneous Localization and Mapping (SLAM), and Augmented Reality (AR), so does other application field[3]. With the arrival of Microsoft's Kinect sensor, its low price, only $200, quickly makes it accessible consumer-level product. In addition, this year's new version of Kinect, Kinect for Windows, and the release of brand new SDK[4] all provide an improved base for individual study.

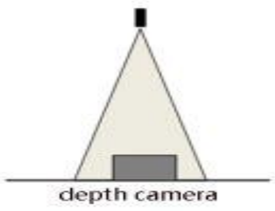

Fig. 3. Principles of depth camera.

B. Kinect Sensor

On June 13, 2010, when an event called the "World Premiere 'Project Natal' for the Xbox 360 Experience" was held by Microsoft, It was announced that the motion sensing device serving as the external equipment of Xbox360 was officially called Kinect. Instead of using a mouse and a keyboard, or any other remote controller, Kinect drastically changed the conventional way people interact with computer by solely depending on capturing the user's motion, allowing them to control the computer or Xbox360 using their own body movements.

As is shown in Fig. 4, Kinect consists of three sensors: an ordinary RGB camera in the middle, an IR light emitter on the left that beams infrared light speckle into space, and a monochrome CMOS sensor on the very right that captures the reflected light. The depth measurement adopted by Kinect sensor is a light-coding technique. Unlike conventional TOF or structure light method, light-coding technique only require an ordinary CMOS sensor rather than a specially made sensor, which drastically lower the cost of the device. In fact, the so-called light-coding technique is still a kind of structure light technique, which code the target environment by the light source. Kinect's IR light emitter can beam a series of infrared light speckle with the shape of the reflected light changing on the basis of the distance. i.e., each shape of the speckle pattern within the environment is unique. All these recognizing and storage work can be finished in a very short time, thus making it possible for Kinect to generate depth map at the rate of 30Hz. Also, its principle of measurement allow it to work under any ambient light conditions.

In February 2012, Microsoft launched a new version of the device, Kinect for Windows. Compared with the old version with Xbox360, the accuracy of depth measurement is improved, a near-mode is added which allow the sensor to measure depth from 0.4m to 3m, all of which make it more suitable for 3D-reconstruction of smaller objects.

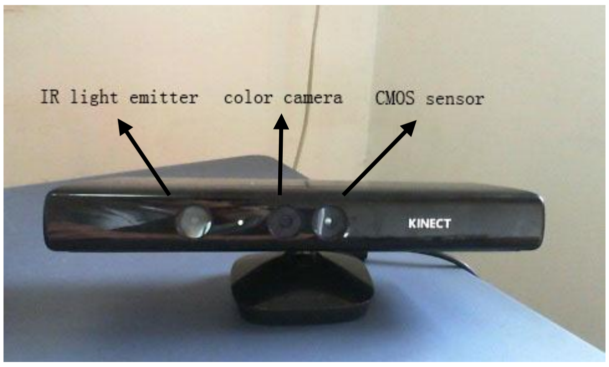

Fig. 4. The Microsoft Kinect sensor.

The standard deviation of depth error and the calibration result of the Kinect (Xbox360 version) are provided by some previous studies[5]-[6], shown in table 1-3 and Fig. 5 where $K = [k_1, k_2, k_3, k_4]$ are the lens distortion parameters.

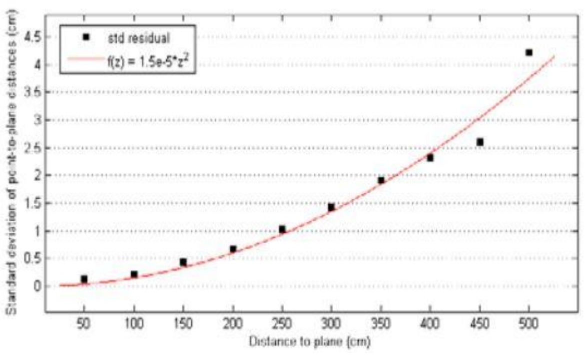

Fig. 5. Standard deviation of plane fitting residuals at different distances of the plane to the sensor.

TABLE I: Color Camera Internal Coefficients

Color internals			
f_{cx}	f_{cy}	u_{c0}	v_{c0}
532.90 ±0.06	531.39 ±0.05	318.57 ±0.07	262.08 ±0.07
k_1	k_2	k_3	k_4
0.2447 ±0.0004	-0.5744 ±0.0017	0.0029 ±0.0001	0.0065 ±0.0001

TABLE II: Depth Camera Internal Coefficients

Depth internals			
f_{dx}	f_{dy}	u_{d0}	v_{d0}
593.36 ±1.81	582.74 ±2.48	322.69 ±1.34	231.48 ±1.59

TABLE III: Relative Pose Between Color and Depth Camera

Relative pose (rad, mm)			
θ_r	t_{rx}	t_{ry}	t_{rz}
0.024 ±0.003	-21.4 ±1.5	0.7 ±1.5	1.0 ±1.9

III. Methods

In this part, we describe the entire workflow of our reconstruction system. Fig. 6 provides an overview of our whole method in block form. It comprise the following four steps:

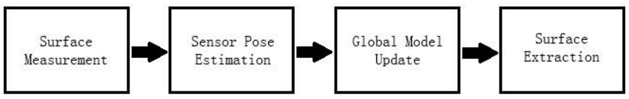

Fig. 6. Overview of the system workflow.

Surface measurement: In this pre-processing stage, first we obtain the raw depth measurements from the Kinect Sensor. After doing some pre-process, we project the data back into the space to generate the dense point-cloud and calculate its corresponding normal map.

Sensor pose estimation: The sensor pose tracking is achieved using ICP alignment [7] between current frame and the previous frame. Then we apply the transform matrix to the point-cloud to get the global coordinate system.

Global model update: By tracking the sensor pose of each frame, the surface measurement is integrated into the global model using a volumetric method describe in[8].

Surface extraction: After fusing all the depth data obtained, we extract the surface from the global model, save it and render it to the screen.

A. *Surface Measurement*

According to our experiments and analysis, we find that the gross error of depth data of Kinect sensor can be categorized into two types. The first will appear when the edges of object surface are scanned, and the second will come when some objects that are not well reflective are scanned, e.g. some metal or glass surface. In addition, small Gaussian noises exist in the depth map, which will make the reconstructed surface unsmooth if they are not reduced.

Gross error when scanning the edge of object surface: Same with LIDAR, the infrared light emitted from Kinect sensor will not be well reflected when encountering some sharp object or the edge of surface. This will result in an unreliable data of the edge, e.g. a smooth linear edge appear to be unsmooth in the depth image obtained from Kinect sensor. When we perform the registration and fusion of the depth data, this kind of gross error will severely harm the result because we cannot assure whether the points near the edge is behind or in front of the surface.

Gross error when scanning objects that are not well reflective: When encountering some specular objects such as metal or glass, e.g. the monitor of computer, Kinect senor can hardly detect the depth data of these surfaces because these types of material do not reflect the infrared light, so no structured-light depth reading was possible. As a result, the depth image contains numerous holes where the data are missing.

Facing the challenges described above, we need to do some pre-process to reduce the noises and eliminate the gross error every time we obtain a raw depth map. We apply a bilateral filter[9] to the raw depth map to reduce the Gaussian noises and eliminate the gross error caused by sharp edges. As to the holes (missing data areas), we solve this problem in the fusion part of our system. After the pre-process, we generate the point-cloud and its corresponding normal map.

1) Bilateral filter

In the bilateral filter, the output pixel value depends on the weighted combination of neighboring pixel value:

$$g(i,j) = \frac{\Sigma_{k,l} f(k,l)\omega(i,j,k,l)}{\Sigma_{k,l}\omega(i,j,k,l)} \tag{1}$$

The weight in the formula above depends upon the domain kernel and the data-dependent range kernel. We can yield the data-dependent bilateral weight function by multiplying the two kernels:

$$\omega(i,j,k,l) = \exp(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2}) \tag{2}$$

In our system, we let the size of window of the filter to be 9-11, σ_d to be 150 and σ_r to be 50.

2) Acquisition of point-cloud and normal

We use a matrix K to represent the camera calibration matrix of the depth camera. At time k, the raw depth map R_k provides the depth measurement $R_k(u)$ at each image pixel **u**. we also represent the camera pose by a transformation matrix $T_{g,k}$ where R and t stand for the 3DOF rotation matrix and the 3DOF translation matrix, respectively.

$$K = \begin{bmatrix} f_{dx} & 0 & u_{d0} \\ 0 & f_{dy} & v_{d0} \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$T_{g,k} = \begin{bmatrix} R_{g,k} & t_{g,k} \\ 0 & 1 \end{bmatrix} \tag{4}$$

$$u = (u,v)^T, \dot{u} = (u^T|1)^T \tag{5}$$

We use $\dot{u}$ to represent the corresponding homogeneous vector of u. The relation between a point $V_k(u) = (x,y,z)^T$ in 3D space and its corresponding image pixel $\dot{\mathbf{u}}$ is：

$$V_k(u) = R_k(u)K^{-1}\dot{u}$$

Now we back-project the filtered depth values into the sensor's scanning space to obtain the point-cloud V_k. Since the depth image from the depth sensor is a surface measurement on a regular grid, we can use a cross product between neighboring map vertices to compute the corresponding normal vectors:

$$N_k(u) = \left(V_k(u+1,v) - V_k(u,v)\right) \times \left(V_k(u,v+1) - Vku,v\right) \tag{6}$$

Then normalize the vector:

$$N_k(u) = N_k(u)/\|N_k(u)\|_2 \tag{7}$$

Another option to obtain the point-cloud directly is to use the OpenNI[10] Library, which can deals with the Microsoft Kinect sensor.

B. *Sensor Pose Estimation*

After we generate the point-cloud, the next step in processing is the registration of partial 3D surface models, i.e. we need to estimate the 3D rigid transformation matrix of the sensor pose and transform the point-cloud into the global system coordinate[11]. Since iteration is needed in the traditional ICP algorithm, which is commonly used in accurate registration, we usually use a feature based registration algorithm to get the initial rough value. For example, optical flow estimation in RGB frame can be used to track point features between two neighboring frames, then the initial relative transformation matrix can be obtained by a RANSAC[12] based alignment, after which the ICP alignment is used to get the accurate sensor pose.

In our system, we utilize the high frame rate of the depth data stream, which result in a high overlap between neighboring frames. So in practice, we simply let the initial matrix to be the identity matrix and perform the frame to frame ICP alignment. After we get the transformation matrix between two neighboring frames, which is called the relative orientation, we compute the global transformation of frame k by multiplying the global transformation $T_{g,k-1}$ of frame k-1 and the relative transformation $T_{r,k}$ between frame k and k-1. Thus the global coordinate of point-cloud generated from frame k can be get.

$$V_{g,k} = T_{g,k} \cdot V_k \tag{8}$$

$$T_{g,k} = T_{g,k-1} \cdot T_{r,k} \tag{9}$$

1) ICP algorithm

The basic principle of ICP algorithm [7] is to find the rotation and translation parameters between two point-clouds or surfaces during iterations.

The rotation parameters is represent by the unit quaternion, which is a four vector as follows:

$$\vec{q_R} = [q_0, q_1, q_2, q_3] \tag{10}$$

where $q_0 \geq 0$ and $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$. The 3×3 rotation matrix can be generated by a unit quaternion as follows:

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_2^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 + q_2^2 - q_1^2 - q_2^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{bmatrix} \tag{11}$$

The translation parameters is:

$$\vec{q_T} = [q_4, q_5, q_6] \tag{12}$$

Thus, the complete registration state vector $\vec{q}$ is:

$$\vec{q} = [\vec{q_R} | \vec{q_T}]^T \tag{13}$$

During the registration, let $P = \{\vec{p_i}\}$ be a measured data point set to be aligned with a model point set $X = \{\vec{x_i}\}$, N_x be the model points amount and N_p be the measured points amount, where $N_x = N_p$ and where each point $\vec{p_i}$ corresponds to the point $\vec{x_i}$ with the same index. The mean square objective function to be minimized is:

$$f(\vec{q}) = \frac{1}{N_p}\sum_{i=1}^{N_p} \|\vec{x_i} - R(\vec{q_R})\vec{p_i} - \vec{q_T}\|^2 \tag{14}$$

The "center mass" $\vec{\mu}_p$ of the measured point set P is given by the average value of all of the measured points p_i, so is the "center mass" $\vec{\mu}_x$ of the model point set X. The cross-covariance matrix of the sets P and X is given by:

$$\Sigma_{px} = \frac{1}{N_p}\sum_{i=1}^{N_p}[(\vec{p_i} - \vec{\mu}_p)(\vec{x_i} - \vec{\mu}_x)^T] = \frac{1}{N_p}\sum_{i=1}^{N_p}[\vec{p_i}\vec{x_i}^T] - \vec{\mu}_p\vec{\mu}_x^{\ T} \tag{15}$$

The cyclic components of the anti-symmetric matrix $A_{i,j}$ are used to form the column vector $\Delta = [A_{2,3} \quad A_{3,1} \quad A_{1,2}]$, this vector is then used to form the symmetric 4×4 matrix $Q(\Sigma_{px})$:

$$Q(\Sigma_{px}) = \begin{bmatrix} tr(\Sigma_{px}) & \Delta^T \\ \Delta & \Sigma_{px} + \Sigma_{px}^{\ T} - tr(\Sigma_{px})I_3 \end{bmatrix} \tag{16}$$

where I_3 is the 3×3 identity matrix. The unit eigenvector $\vec{q_R} = [q_0, q_1, q_2, q_3]$ corresponding to the maximum eigenvalue of the matrix above is selected as the optimal rotation parameters. Then the translation vector is given by: $\vec{q_T} = \vec{\mu}_x - R(\vec{q_R})\vec{\mu}_p$, after which we can get the complete registration state vector $\vec{q}$.

Next, we apply the transformation to the measured point data set and compute the mean square distance $f(\vec{q})$. The whole steps described above is repeated until the mean-square error falls below a preset threshold.

2) Global optimization

Accumulation of errors in the frame to frame alignment can be gradually noticeable when the camera has moved a relatively long distance from its original place. This will result in a reconstruction that has two representation in the same area in different locations. We need to solve this problem by a global optimization.

Suppose we have n aligned depth frame, first, we detect the loop closure automatically through point features matching. Then the global transformation is optimized using a least-square function:

$$F = V^T P V \tag{17}$$

$$\begin{cases} V = [D_1 \quad D_2 \quad \cdots \quad D_n] \\ D_k = [D_{k,1} \quad D_{k,2} \quad \cdots \quad D_{k,p}] \\ D_{k,i} = \|T_{g,k}R_{k,i} - T_{g,k-1}R_{k-1,i}\|_2 \end{cases} \tag{18}$$

where $R_{k,i}$ and $R_{k-1,i}$ are the corresponding points in the *k*th and *k-1*th depth frame obtained in the ICP alignment. For the first frame, the Euclidean distance is computed between the first and last frame, i.e. $D_{1,i} = \|T_{g,1}R_{1,i} - T_{g,n}R_{n,i}\|_2$. By minimizing F, the distance between corresponding points will become minimal.

C. Global Model Update

With the sensor pose estimated, each consecutive depth frame is fused incrementally into one single surface model using a volumetric method called truncated signed distance function[8]. Utilizing the high frame rate of the depth data stream, we can make use of all of the depth data, so the system can gradually fill the holes cause by some materials that does not reflect infrared light well.

1) *Volumetric representation*

A voxel (volumetric pixel) is a volume element, representing a value on a regular grid in 3D space. This is analogous to a pixel, which represents 2D image data in a digital picture. As with pixels in a digital picture, voxels only have the relative position based upon other voxels' position. In contrast, points or polygons are often explicitly represented by the coordinates of their vertices.

Shown in Fig. 7, a voxel represents a data point in 3D space. This data point can contain a single piece of data, such as color, as well as multiple piece of data, such as opacity, color and normal vector. Depending on the application area and the intended use for the data, the value of a voxel may represent various property. Also, value of points among voxels can be get via interpolation.

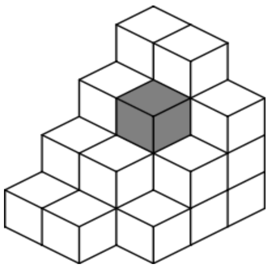

Fig. 7. Volumetric representation of 3D space.

With a resolution of 500×500×500, we choose 2mm as the side of length of each voxel in our system due to the size of reconstructed object, i.e. the whole space is 1m^3.

2) *Truncated signed distance function*

Article[8] present a weighted signed distance function D(x) as follows:

$$D(x) = \frac{\sum w_i(x) d_i(x)}{\sum w_i(x)} \tag{19}$$

$$W(x) = \sum w_i(x) \tag{20}$$

where x represent a point in 3D space, $d_i(x)$ is the signed distance function from the *i*th range image and $w_i(x)$ is the weight function. The signed distance mentioned above is the distance of each point **x** to the nearest range surface along the line of sight to the sensor. Combining the two functions $d_i(x)$ and $w_i(x)$, we can construct the weighted signed distance function, which gives us each voxel a cumulative signed distance function, $D(x)$, and a cumulative weight $W(x)$. For example, in our system, each voxel value contains a distance and weight, i.e. the distance to the nearest surface and its corresponding weight. We can extract the 3D surface model by solving $D(x) = 0$.

Fig. 8 illustrates the principle of unweighted signed distance function. A range sensor looking down the x-axis observes a range image. Following one line of sight down the x-axis, the space in front of the surface has a negative distance while the behind has a positive distance. Note that when the weight is 1, the resulting surface would be the surface created by averaging the two range surfaces along the sensor's lines of sight. In general, the weights should be specific to the range sensor, e.g. we can make the weight depend on the dot product between each vertex normal and the viewing direction for some range sensor[8].

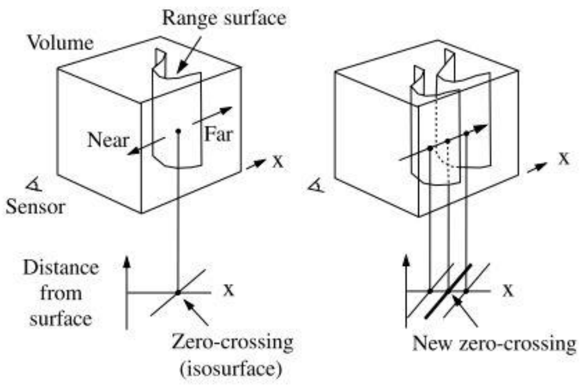

Fig. 8. Principle of unweighted signed distance function.

For each depth frame with the absolute orientation estimated, we fuse them incrementally following the rules as follows:

$$D_{i+1}(x) = \frac{W_i(x)D_i(x) + w_{i+1}(x)d_{i+1}(x)}{W_i(x) + w_{i+1}(x)} \tag{21}$$

$$W_{i+1}(x) = W_i(x) + w_{i+1}(x) \tag{22}$$

where $D_i(x)$ and $W_i(x)$ are the cumulative signed distance functions and weight functions after fusing the *i*th range image. In principle, the distance and weighting functions should extend indefinitely in either direction. However, to prevent surfaces on opposite sides of the object from interfering with each other, we force the weight to taper off behind the surface. i.e. the distance functions should be truncated at a certain distance, e.g. the half the maximum uncertainty interval of the range measurements as described in [8]. In practice, we also just let $W_{R_k}(x) = 1$, which can provides a good results in the experiments.

When computing the signed distance, we use a projective truncated signed distance function[13] that is trivially parallelizable, which can boost our system efficiently in the future if we take advantages of GPU programming. For a raw depth map R_k, the signed distance at point **p** is computed as follows:

$$F_{R_k}(p) = \psi\left(\lambda^{-1}\|(t_{g,\mathrm{k}} - \mathrm{p})\|_2 - R_k(\mathrm{x})\right) \tag{23}$$

$$\lambda = \|K^{-1}\dot{x}\|_2 \tag{24}$$

$$\mathrm{x} = \lfloor \pi(\mathrm{K}T_{g,\mathrm{k}}^{-1}\mathrm{p}) \rfloor \tag{25}$$

$$\psi(\eta) = \begin{cases} min\left(1, \frac{\eta}{\mu}\right) sgn(\eta), \ \eta \geq -\mu \\ \qquad\qquad \text{null}, \ otherwise \end{cases} \tag{26}$$

where K is the depth camera calibration matrix, T is the camera pose matrix, R_k is the depth image at time k, $\pi(x)$ performs perspective projection of x, λ^{-1} converts the ray distance to p to a depth and $\psi(\eta)$ performs the SDF truncation. In order to prevent smearing of measurements at depth discontinuities, we use the nearest neighbor lookup ⌊x⌋ instead of interpolating the depth value[13].

Now we summarize the whole algorithm used in our system as follows: First, we apply the inverse of global transformation matrix of frame k to the volumetric grid to transform its coordinate to current frame coordinate. Next, we project each voxel p onto the image plane of the depth sensor to get its corresponding image pixel x using the nearest neighbor lookup method. Then, we compute signed distance by calculating the difference between the depth of each voxel p and the depth $R_k(\mathrm{x})$ of its corresponding image pixel x. Finally, we perform the truncation of the signed distance.

D. Surface Extraction

It is obvious that the distance between the point on the surface and the surface is zero. So we can extract the surface by finding the points that satisfy $F(x) = 0$. Given the SDF representation, two main approaches to obtaining the surface have been extensively studied within the graphics community. One is the marching cubes algorithm[14]. Alternatively, the surface can be directly raycast using the algorithm described in[15]. Advantage of the former option is that we can obtain the whole 3D reconstruction result of the scanning scene, which is useful if we want to build a real 3D model of the objects, such as a human head. On the other hand, the latter option can avoid the need to visit areas of the function that are outside the desired view frustum. This is necessary in the sensor pose estimation part of the reconstruction workflow if we want to perform the frame to global registration.

In our system, we use an option different from the approaches described above to obtain the surface. Every time we update the global TSDF, each voxel is projected onto the image plane of the depth sensor to get its corresponding image pixel $\mathbf{x} = \lfloor \pi\left(\mathrm{K}T_{g,\mathrm{k}}^{-1}\mathrm{p}\right) \rfloor$ then the signed distance is computed. At the same time, with the normal map computed in the pre-process stage, we use the normal of $\mathbf{x}$ to represent the normal of the projected voxel using the similar formula as follows:

$$N(x) = \frac{W_i(x)N_i(x) + w_{i+1}(x)n_{i+1}(x)}{W_i(x) + w_{i+1}(x)} \qquad (27)$$

For a voxel representing point $\mathbf{p}$, the surface it contains is:

$$S(p) = p + N(p) \cdot \mathrm{F}(p) \qquad (28)$$

However, if the surface extracted is outside the voxel, we do not accept it.

IV. Experiment

Kinect for Windows SDK[4], OpenCV library[16] and Point Cloud Library[17] is used for implementing the algorithm described in this paper. First, we obtain the raw depth map from Kinect sensor using Kinect SDK, then apply the bilateral filter function in OpenCV to the raw depth map, next we generate the point-cloud and normal map from the filtered depth map as shown in Fig. 9-10.

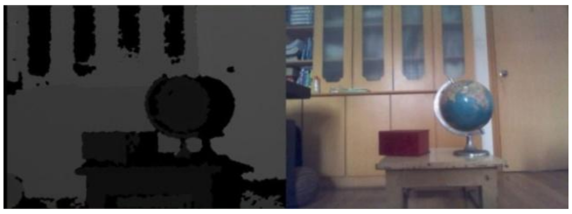
Fig. 9. Raw depth map and color image

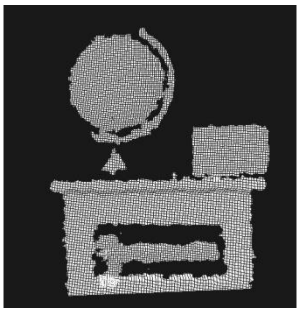
Fig. 10. Point-cloud obtained from raw depth map

Next we perform the registration between neighboring frames using ICP algorithm. Then, with the camera pose of each frame estimated, we fuse the depth data using TSDF. Fig. 11 presents the comparison between the point-cloud generated from raw depth map and from the fused model.

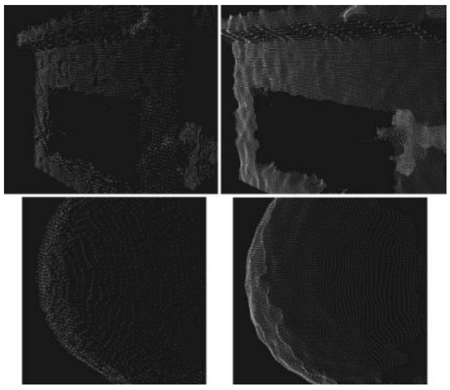
Fig. 11. Point-clouds generated from raw depth map(right) and from fused model(left)

It is apparent that the fused model is much smoother than the original one.

Finally, the complete rendered model is presented as follows:

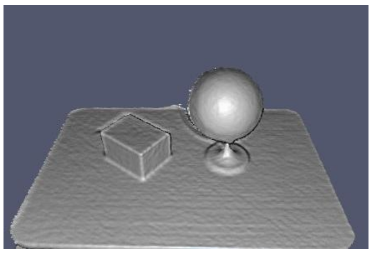
Fig. 12. The complete model.

V. Conclusion

With the arrival of Microsoft Kinect sensor, its huge market potential will open up many new possibilities for augmented reality, human-computer-interaction and other field. In this paper, we present a workflow to reconstruct small 3D objects. Four major steps in our system are: 1>Data acquisition from Kinect sensor and pre-process; 2>Registration of the point-cloud to get the camera pose; 3>Integration of the depth data using a volumetric method; 4>3D surface extraction.

There are several ways in which our system could be improved.

A. Frame-Model Registration

The frame to frame registration strategy used in our system will result in a rapid accumulation of errors and a poor 3D reconstruction quality, because the accuracy of pose estimation is affected by the pose of previous frame. Thus a circular trajectory of the sensor is needed and we detect the loop closure to perform a global optimization[18].

However, the frame to model registration strategy[13] does not require explicit global optimization since the loop is closed between mapping and localization by tracking the live depth frame against the global TSDF model. Utilizing the ray cast method[15], we can provide a dense surface prediction against which the live depth map is aligned.

B. Parallel Processing

Currently, research on simultaneous localization and mapping (SLAM) has focused more on real-time tracking and reconstruction. Parallelizing the process is the essential ideal for real-time reconstruction. Taking advantage of GPGPU processing hardware, we can easily parallelize the algorithms used in our system, such the registration and fusion part. For example, A GPU-Accelerated nearest neighbor search method[19] can be used in our ICP alignment algorithm. Moreover, NVIDIA's CUDA[20] and the GPU version function of OpenCV[16] are also helpful in enhancing the performance of our system.

C. Large-Scale Reconstruction

The current system only works well for reconstructing small size objects with volumes of $< 5m^3$. Several challenging problems exist if we want to reconstruct large-scaled models such as the interior scene of a building. One is that the current volumetric representation would require too much computer memory. The other is how to perform automatic relocalization when the tracking has failed.

ACKNOWLEDGMENT

Thanks to our roommates for your help in our experiment. Thanks to Wuhan University for offering us a good environment for study and experiment. Thanks to the Microsoft for developing such a revolutionary product.

REFERENCES

[1] Z. Zhang, "Digital photogrammetry and computer vision," *Geomatics and Information Science of Wuhan University*, vol. 29, no. 12, pp. 1035-1039, 2004.

[2] Y. Feng. *Close Range Photogrammetry*. Wuhan: Wuhan University, 2002, pp. 7-8.

[3] M. Liao, Q. Zhang, H. Wang, R. Yang, and M. Gong, "Modeling deformable objects from a single depth camera," *in Proceedings of the International Conference on Computer Vision*, 2009.

[4] Kinect for Windows SDK Documentation (2012). [Online]. Available: http://msdn.microsoft.com/en-us/library/hh855347.aspx/

[5] C. D. Herrera, J. Kannala, and J. Heikkila, "Accurate and practical calibration of a depth and color camera pair," in *Proceedings of Computer Analysis of Images and Patterns*, 2011.

[6] K. Khoshelham, "Accuracy analysis of kinect depth data," in *Proceedings of ISPRS Workshop Laser Scanning*, 2011.

[7] P. J. Besl and H. D. M. Kay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239 - 256, 1992.

[8] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1996.

[9] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of International Conference on Computer Vision*, 1998.

[10] OpenNI (Open Natural Interaction) Programmer Guide. [Online]. Available: http://openni.org/Documentation/ProgrammerGuide.html, 2012.

[11] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010, pp. 588-589.

[12] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381-395, 1981.

[13] R. A. Newcombe, S. Izadi, O. Hilliges *et al.*, "Kinect fusion: Real-time dense surface mapping and tracking," *in Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[14] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1987.

[15] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan, "Interactive ray tracing for isosurface rendering," in *Proceedings of the Conference on Visualization*, 1998.

[16] G. Bradski. The OpenCV Library. *Dr. Dobbs Journal of Software Tools*, Nov 2000, pp. 120–126.

[17] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '11)*, Shanghai, China, May 2011.

[18] P. Henry, M. Krainin, E. Herbst *et al.*, "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments," in *Proceedings of the International Symposium on Experimental Robotics*, 2010.

[19] D. Qiu, S. May, and A. Nüchter, "GPU-accelerated nearest neighbor search for 3D registration," in *Proceedings of the International Conference on Computer Vision Systems*, 2009.

[20] NVIDIA CUDA (Compute Unified Device Architecture). Programming Guide. [Online]. Available: http://docs.nvidia.com/cuda/index.html, 2012.

T. Song was born in Nanjing, at December 9, 1990. He will get a Bachelor of Engineering degree in Photogrammetry at Wuhan University in June, 2013. He is now with the School of Remote Sensing and Information Engineering, Wuhan University, Wuhan, Hubei 430070 China. His area of interests are Computer Programming, Close-Range Photogrammetry, Image Processing, Computer Vision and Software Engineering.