

Research Article

Marching Cubes Algorithm for Fast 3D Modeling of Human Face by Incremental Data Fusion

Xiangsheng Huang, Xinghao Chen, Tao Tang, and Ziling Huang

Institute of Automation, Chinese Academy of Sciences, Beijing 100090, China

Correspondence should be addressed to Xiangsheng Huang; xiangsheng.huang@ia.ac.cn

Received 22 December 2012; Accepted 20 January 2013

Academic Editor: Sheng-Yong Chen

Copyright © 2013 Xiangsheng Huang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We present a 3D reconstruction system to realize fast 3D modeling using a vision sensor. The system can automatically detect the face region and obtain the depth data as well as color image data once a person appears in front of the sensor. When the user rotates his head around, the system will track the pose and integrate the new data incrementally to obtain a complete model of the personal head quickly. In the system, iterative closest point (ICP) algorithm is first used to track the pose of the head, and then a volumetric integration method is used to fuse all the data obtained. Third, ray casting algorithm extracts the final vertices of the model, and finally marching cubes algorithm generates the polygonal mesh of the reconstructed face model for displaying. During the process, we also make improvements to speed up the system for human face reconstruction. The system is very convenient for real-world applications, since it can run very quickly and be easily operated.

1. Introduction

3D reconstruction has always been an interesting topic since Microsoft Kinect camera came to use. The depth data of human face can be easily acquired using depth camera. However, it is still difficult to get a perfect whole face model. One probable method to solve this problem is to obtain depth maps from different cameras in different directions simultaneously [1–5]. Another solution is to get data with only one camera in different time. The next step is to do reconstruction with all the data from different directions, finally generating a mesh model of human face. A lot of algorithms for reconstruction have been proposed recently but hardly get perfect results. Moreover, many of them are not so conveniently or easily used in practice.

In this paper, we present a system that realizes fast human face tracking and 3D reconstruction with only one Kinect sensor collecting depth data (Figure 1). The depth data is acquired from the Kinect sensor and is converted to vertex map and normal map, as there is a corresponding relation between the 2D depth map coordinate system and the 3D camera coordinate system. The ICP algorithm is used to track the relative pose between the current data map and the

previous one, and then the transformation matrix frame-to-frame can be estimated. Thereby the pose of current camera in the global coordinate can be obtained, which is actually the human face coordinate system. By doing the volumetric integrating using truncated signed distance function, the depth data of all the frames can be integrated into a volume, and the zero-crossing surface can be extracted for the next ICP tracking. The reconstructed 3D model is rendered using the marching cubes algorithm in the reconstructing process, so the user can adjust his face pose to get a better human face model by filling the hole. This system allows users to reconstruct human face model with only one depth camera and incrementally get higher quality 3D human face model by rotating the head.

2. Related Works

The research on real-time 3D reconstruction is a hot topic in computer vision. An accurate and robust 3D surface registration method for In-hand modeling was proposed by [6]. A complete 3D model of an object can be obtained by simply turning around and being scanned by a camera, using iterative closest points (ICPs) algorithm [1] for coarse and



FIGURE 1: User sits in front of a fixed Kinect sensor and the reconstruction can be done.

fine registration. The authors also proposed a method for detecting registration failure based on both geometric and texture consistencies [6]. With the user performing slight head rotation while keeping the facial expression unchanged, the system proposed by Weise et al. [7] aggregated multiple scans into one 3D model of the face. A method for automatically registering multiple 3D data sets without any knowledge of initial pose was proposed by [8]. Jaeggli et al. [9] presented a system which produces complete 3D model using a high-speed acquisition equipment and a registration module. They used pairwise registration as well as multiview refinement to get better results. Azernikov and Fischer [10] proposed volume warping method for surface reconstruction.

KinectFusion project [11–13] presented a system that uses only one moving depth camera to create detailed 3D reconstruction of a complex and arbitrary indoor scenes in real time with GPU. It also enables advanced augmented reality and multitouch on any indoor scene with arbitrary surface geometries. Zollhöfer et al. [14] proposed an algorithm for computing a personalized avatar using a single color image and corresponding depth image. It obtains a high-quality 3D reconstruction model of the face that has one-to-one corresponding geometry and texture with the generic face model.

3. Method and Implementation

The flow chart of our 3D face reconstruction system is shown in Figure 2. The whole system mainly consists of six stages: face detection and segmentation, depth map conversion, face tracking, volumetric integration, recasting, and marching cubes. Each stage will be described in the following sections.

3.1. Face Detection and Segmentation. The Kinect sensor acquires the 640×480 color and depth image at 30 Hz. Firstly the face region is detected using a Haar classifier [15]. To get more stable results, we use the frontal-face and profile-face classifiers to detect the face twice.

A more sophisticated extraction is used to ensure that only the face data in consideration is carried out after getting the region of the face. We search the depth image in a window around the central point of the face region to get a valid

depth value. Then the depth image will be traversed within the face region and every depth value will be compared with the central depth value. If the depth value changes more than the specific threshold, it will be given an invalid value to distinguish nonface region from face region.

In order to run the algorithm fast, the face region is only detected at the beginning of the algorithm or after resetting. Once a valid bounding rectangle of face is obtained, the face detection phase is omitted and only segmentation task is executed.

3.2. Fast Face Depth Map Conversion. Firstly, a bilateral filter is applied to the raw depth map in order to obtain noise-reduced face depth map and maintain the depth boundaries simultaneously. The filter can be described as follows [12], (raw map is denoted by R_i , and the filtered depth map is denoted by D_i):

$$D_i(\vec{u}) = \frac{1}{W_p} \sum_{\vec{q} \in w} N_\sigma(\|\vec{u} - \vec{q}\|_2) N_\sigma(\|R_i(\vec{u}) - R_i(\vec{q})\|_2) R_i(\vec{q}), \quad (1)$$

where $\vec{u} = (u, v)^T$ is the depth image pixel and $\vec{q} \in w$ (w is a window to reduce computation complexity), W_p and $N_\sigma(t) = \exp(-t^2 \sigma^{-2})$ are normalizing constants.

In a real-time system, the time complexity is an important factor. In the filter previously mentioned, the exponent arithmetic is computationally expensive. Since the distance between the depth image pixels is an integer, we can use a *lookup* table to speed up the computation. Only the pixels within a certain window are considered, so the size of lookup table is not large.

Given the camera intrinsic parameters (f_x, f_y, c_x, c_y , which, respectively, stand for the focal length in x and y axes, the focal point coordinate in x and y axes), the depth map can be converted into the vertex map (denoted by V_i), and corresponding normal vectors for each vertex can be easily computed using neighboring points.

Assuming that (u, v) is a pixel in the raw depth map, and $z = D_i(u, v)$ is the depth of filtered depth map, then the coordinate of V_i can be computed as follows:

$$\begin{aligned} V_i(x) &= \frac{z(u - c_x)}{f_x}, \\ V_i(y) &= \frac{z(u - c_y)}{f_y}, \\ V_i(z) &= z. \end{aligned} \quad (2)$$

With the vertex data obtained, the normal vectors of each vertex are computed with the following equation [12]:

$$N_i = (V_i(u + 1, v) - V_i(u, v)) \times (V_i(u, v + 1) - V_i(u, v)), \quad (3)$$

and then normalized to the unit length using $N_i / \|N_i\|$.

Assuming that the image being processed has $m \times n$ pixels, and the window of bilateral filter has a size of $w \times w$, the

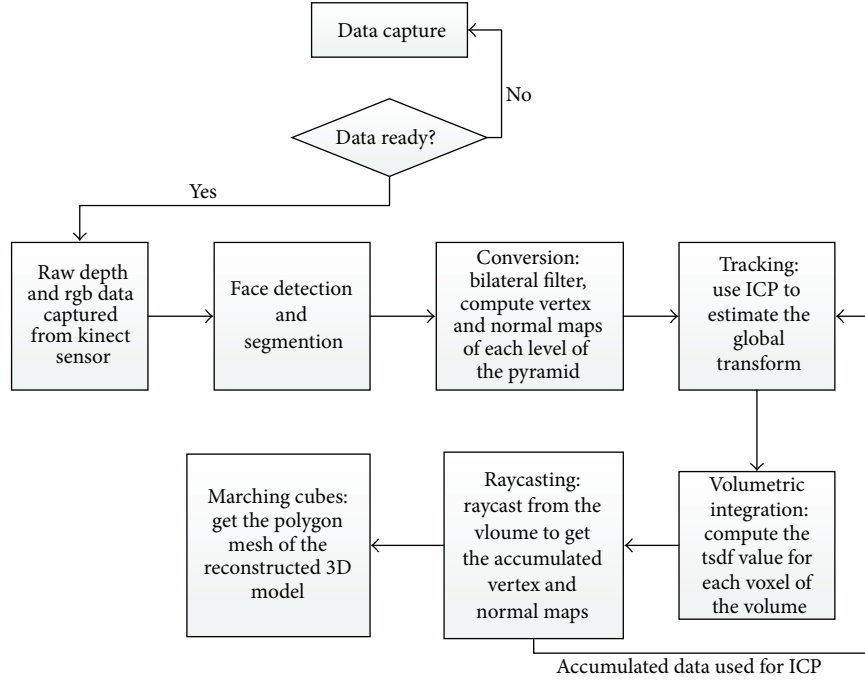


FIGURE 2: Overall system workflow.

computational complexity of filtering is $O(m \times n \times w^2)$. The Kinect sensor acquires a 640×640 depth image. In the face segmentation stage, a smaller image that contains the face region is extracted, which has a size of 196×196 in our experiment. Besides, $w = 7$ and a lookup table is used to replace the exponent arithmetic. A complexity of $O(m \times n)$ is needed to generate the normal vectors. So the computational complexity of this stage is quite low.

3.3. Face Tracking. The global model of the face is stored as a statue, which means that the vertex map and the normal map do not move and rotate in the coordinate. In the meantime, the sensor is fixed in the front of the user. However, because the user's head rotates (and moves) in the modeling process, the view point of the depth data changes from frame to frame. In order to integrate the history model and the live data, the transformation matrix (a 3D matrix with 6 degrees of freedom) between them is required; otherwise the result will be chaos if we put the data directly. Fortunately, because of the preextraction, the live data do not contain environment and other references. So we come out with the solution that we suppose that the camera is moving around to scan a static head of the user. In this stage, the face depth data is processed frame-to-frame to track the face motion. The Iterative Closest Point (ICP) algorithm [1] is used here to compute the transformation matrix [12].

In ICP algorithm, each iteration could be divided into three stages: first, corresponding point pairs are chosen within the two frames; then a point-to-plane Euclid distance error metric is used to calculate an optimal solution which minimizes the error metric; and finally, the previous frame is transformed using the matrix obtained in the previous

stage, preparing for the next iteration. Solution becomes more accurate after each cycle, and an optimal transform matrix can be obtained after finishing ICP algorithm.

In the first stage, the corresponding points between the current frame and the previous frame should be found firstly. In this paper, points in the current frame are projected to the previous one to obtain the corresponding points. Given the previous global camera pose T_{i-1} and the estimated current pose T_i (which is initialized with T_{i-1} before the iteration and is updated with an incremental transform calculated per iteration of ICP), we then transform the current vertex map into the global coordinate, using $V_i^g = T_i V_i^c(\vec{u})$. (Here, the human face coordinate system is regarded as the global system. Since our target is to obtain a complete face model, so that the data is required in face coordinate system.) The pixel corresponding to the global position V_i^g are required here, so we transform V_i^g into the previous camera coordinate system to get $V_{i-1}^c = T_{i-1}^{-1} V_i^g$ and then project it into the image plane to get the pixel p :

$$\begin{aligned}
 p(u) &= \frac{V_{i-1}^c(x) \times f_x}{V_{i-1}^c(z)} + c_x, \\
 p(v) &= \frac{V_{i-1}^c(y) \times f_y}{V_{i-1}^c(z)} + c_y.
 \end{aligned} \tag{4}$$

We look up the previous global vertex map V_{i-1}^g and global normal map N_{i-1}^g in the pixel p ; obviously $V_{i-1}(\vec{p})$ and $V_i(\vec{u})$ are the correspondences.

Also thresholds of Euclidean distance and angles should be set between them, to reject outliers.

Given these sets of corresponding oriented points, the next step is to minimize the point-to-plane error metric E , to get the transformation matrix T [16]. Here, the metric E stands for the sum of squared distances between the current points and the tangent plane of the corresponding points:

$$E = \sum_{\vec{u}} \|(TV_i(\vec{u}) - V_{i-1}^g(\vec{p})) N_{i-1}^g(\vec{p})\|_2. \quad (5)$$

As there is only an incremental transformation between frames, the rigid transform matrix can be written as follows:

$$T = [R | \vec{t}] = \begin{pmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \end{pmatrix}. \quad (6)$$

In each cycle, an incremental transformation T_{inc}^z (z represents the current iteration number) that minimizes the error metric can be estimated. The desired global transformation matrix can be simply updated by $T_g^z = T_{\text{inc}}^z \cdot T_g^{z-1}$.

We update the current global frame vertex estimates using the global transformation T_g^{z-1} computed in the previous iteration, $V_i^g(\vec{u}) = T_g^{z-1} V_i(\vec{u})$. The increment transformation can also be written as follow:

$$\vec{x} = (\beta, \gamma, \alpha, t_x, t_y, t_z) \in \mathcal{R}^6, \quad (7)$$

$$T_g^z V_i(\vec{u}) = R^z V_i^g(\vec{u}) + \vec{t}^z = G(\vec{u}) \vec{x} + V_i^g(\vec{u}).$$

Assuming that $V_i^g(\vec{u}) = (x, y, z)^T$, G can be represented as follows:

$$G(\vec{u}) = \begin{pmatrix} 0 & -z & y & 1 & 0 & 0 \\ z & 0 & -x & 0 & 1 & 0 \\ -y & x & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (8)$$

By solving the following expression:

$$\min_{\vec{x} \in \mathcal{R}^6} \sum_{\vec{u}} \|E\|_2, \quad (9)$$

$$E = N_{i-1}^g(\vec{p})^T \cdot (G(\vec{u}) \vec{x} + V_i^g(\vec{u}) - V_{i-1}^g(\vec{p})).$$

An optimal transformation matrix needed can be computed.

Then with the equations previously mentioned, the expression can be expressed as follows:

$$\sum_{\vec{u}} (A^T A) \vec{x} = \sum_{\vec{u}} A^T b, \quad (10)$$

where $A^T = G^T(\vec{u}) N_{i-1}^g(\vec{p}) \in \mathcal{R}^{6 \times 1}$ and $b = N_{i-1}^g(\vec{p})^T (V_{i-1}^g(\vec{p}) - V_i^g(\vec{u})) \in \mathcal{R}^{1 \times 1}$, and we can easily compute the vector \vec{x} using a Cholesky decomposition.

After the incremental matrix is obtained, we enter the next iteration, where the operations previously mentioned are used again. And after all the iterations, we can get the final camera pose $T_g \leftarrow T_g^{z_{\text{max}}}$.

A projecting strategy is used so that it takes $O(1)$ computational complexity for a point to find its corresponding point. To get the transformation matrix between two frames, approximately a computational complexity of $O(m \times n)$ is acquired.

3.4. Small Size Face Volumetric Integration. Using the transformation matrix obtained in the last step, the depth map data can be converted to point cloud in global coordinate. Data of this form is hard to integrate to form a face model. Here we convert them into a volumetric representation [17]. A 3D volume of specific resolution that corresponds to the physical space is divided into a 3D grid of voxels. Truncated signed distance function (TSDF) is used here to convert the 3D vertices into voxels of the volume [12].

In the volumetric integration phase, each voxel in the volume is traversed and the corresponding TSDF value is updated using weighted average strategy. For a voxel (x, y, z) in the volume, we first convert it into the global 3D position v_g :

$$\begin{aligned} v_g(x) &= (x + 0.5f) \times \text{cell_size.x}, \\ v_g(y) &= (y + 0.5f) \times \text{cell_size.y}, \\ v_g(z) &= (z + 0.5f) \times \text{cell_size.z}, \end{aligned} \quad (11)$$

where cell_size represents the size of the cell in the volume:

$$\text{cell_size.x} = \frac{\text{VOLUME_SIZE_X}}{\text{VOLUME_X}}, \quad (12)$$

where VOLUME_X indicates how many cells are there in x axis of the volume and VOLUME_SIZE_X indicates the corresponding actual length.

Subsequently, the global ordinate v_g is transformed into the camera ordinate v , and the vertex v is projected into the image plane to get the corresponding pixel p .

Assuming that the translation vector of the global camera transformation is denoted as t_i , the distance between voxel (x, y, z) of the volume and the original point of the camera ordinates system can be calculated as $\|v_g - t_i\|$. Since we have got the corresponding pixel p , we can get the actual depth measurement $D_i(p)$. It should be pointed out that $D_i(p)$ is not equal to the vertex map $V_i(p)$, since the former represents the distance between the original point and the specific point, while the latter only represents the z value, so a conversion is necessary to get $D_i(p)$.

The SDF value of the voxel can be computed using $\text{SDF}_i = \|v_g - t_i\| - D_i(p)$. This is normalized to a TSDF using the truncating strategy. The TSDF value of the voxel is updated using a simple running weighted average:

$$\text{tsdf}^{\text{avg}} = \frac{\text{tsdf}_{i-1} \cdot w_{i-1} + \text{tsdf}_i \cdot w_i}{w_{i-1} + w_i}. \quad (13)$$

Actually, in practice, we simply let $w_i = 1$ and can achieve good results.

Given that the goal of our work is fast tracking and reconstruction, time complexity must be considered. In volumetric integration phase, there are $\text{VOLUME_X} \times \text{VOLUME_Y} \times \text{VOLUME_Z}$ voxels that should be traversed, so the volume size cannot be too large for higher frame rate. In case of face reconstruction, the useful global size is approximately $0.3 \times 0.3 \times 0.3 \text{ m}^3$. However, in the volumetric integrating algorithm described previously, the $z = 0$ voxel of the volume

lies in the original position of the camera ordinate system. In other words, if the distance between the face and the Kinect sensor is 0.7 m, $VOLUME_SIZE_Z$ cannot be less than 0.7 to ensure valid integration. And at the same time, all the $0 < z < 0.7$ voxels are not used. Consequently, to ensure enough volume resolution, $VOLUME_Z$ cannot be too small, which could result in higher time complexity and do too much useless work.

Thus we introduce an offset to the volume. We move the volume along the z axis a distance of offset as the $z = 0$ plane can get close to but cannot reach the mesh of the face. Then the conversion from voxel into global position should be modified as

$$v_g(z) = (z + 0.5f) \times cell_size.z + offset. \quad (14)$$

Some little modification should also be made in the other part of the algorithm.

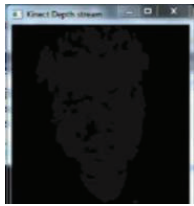

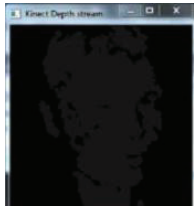

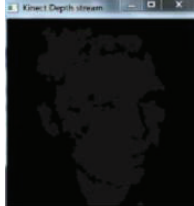

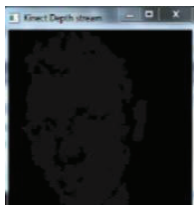
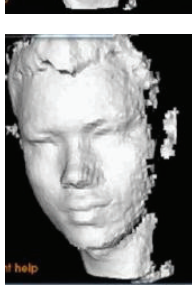
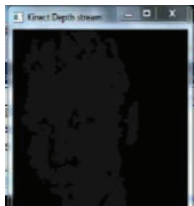
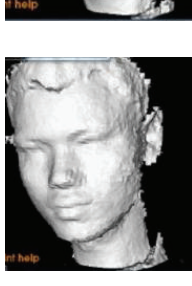
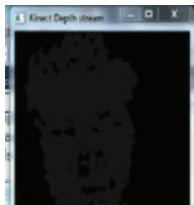
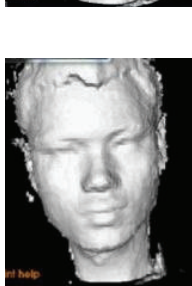
In this stage, a computational complexity of $O(n^3)$ (n is the length of the volume) is needed to update all the voxels in the volume. In our experiment, we use $64 \times 64 \times 64$ ($n = 64$ volume and $0.3 \times 0.3 \times 0.3$ volume size along with an offset of 0.6 m and get rather good fast reconstruction results.

3.5. Ray Casting. The ray casting algorithm [18] applied here is to generate views of implicit surface for rendering and tracking. For each pixel (x, y) of the output image, a single ray is emitted from the original point of the camera coordinate system and goes through the point (x, y) of the image plane. With the direction of the ray, we can extract the surface position by traversing along the ray. And the surface intersection point can be easily obtained using linear interpolation. Then we can easily obtain the normal map with TSDF. There are two contributions of the ray casting algorithm, one is the ability to view the implicit surface of the reconstructed 3D model and the other is to generate higher quality data for ICP camera tracking. When rendered to screen, the noise, shadows, and holes will be much less than the raw depth data.

We need to traverse all the voxels in the volume to extract the zero-crossing surface. Therefore a computational complexity of $O(n^3)$ (n is the length of the volume) is acquired.

3.6. Marching Cubes. In our work, we use marching cubes algorithm [19, 20] to obtain the mesh of the reconstructed 3D model. Each voxel of the volume is traversed and an index is created to a precalculated array of 256 possible polygon configurations within the cube, by treating each of the 8 scalar values as a bit in an 8-bit integer. If the scalar's value is higher than the iso value (it is inside the surface) then the appropriate bit is set to one, while if it is lower (outside), it is set to zero. The final value after all 8 scalars are checked is the actual index to the polygon indices array. Finally each vertex of the generated polygons is placed on the appropriate position along the cube's edge by linearly interpolating the two scalar values that are connected by that edge.

TABLE 1: Results of human face model reconstruction.

Raw depth	Reconstructed mesh	Time
		Shortly after the start of the reconstruction
		Begin to scan the right side of the face
		The right side of the face has almost been reconstructed
		Begin to scan the left side of the face
		The left side of the face has almost been reconstructed
		Reconstruction done

In marching cubes algorithm, the computational complexity that is approximately $O(n^3)$ (n is the length of the volume) is acquired.

4. Results

We test our 3D reconstruction system on a computer with 3.2 GHz CPU and 4 GB memory. We set the volume resolution to $64 \times 64 \times 64$ and the volumetric size to $0.3 \times 0.3 \times 0.3 \text{ m}^3$ with an offset distance of 0.4 m. Note that running the reconstruction algorithm for one new frame only costs about 180ms, which is quite acceptable in practice.

The results of our 3D reconstruction system are shown in Table 1. As shown in the table, the 3D face model keeps being refined, while the user's head rotates in order to let the Kinect scan the whole face.

We can find that the reconstruction result is very good, and is much smoother than the raw depth data. And the reconstruction speed is also very acceptable.

5. Conclusions

After depth cameras like Kinect sensor appear, users can easily obtain the depth data of an object. 3D reconstruction, especially human face reconstruction, has always been a challenging problem. In this paper, we represent a novel way to create a 3D face reconstruction model. Just sitting in front of the Kinect camera and rotating his head, the user can get a perfect human face model. We use a volumetric integrating strategy to fuse all the data, so the reconstructed face model becomes more and more clear.

We contribute the method to fast human face 3D reconstruction. Our efforts to speed up the system are threefold. First, we decrease the frequency of detecting face, by only detecting when the shift of the face exceeds a specific threshold. Second, we use a lookup table to replace the computationally expensive exponent arithmetic and try hard to reduce repeated computation. Third, we introduce some variances to the volumetric integration algorithm to use less voxels while keeping the good resolution. Using the methods previously mentioned, we get a well-performed face 3D reconstruction system.

We will focus on larger object such as full body 3D reconstruction, and add color information to the model to make the visualization better in the future work.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (NSFC) under the Project 61175034/F030410.

References

- [1] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.
- [2] C. Shengyong, W. Yuehui, and C. Carlo, "Key issues in modeling of complex 3D structures from video sequences," *Mathematical Problems in Engineering*, vol. 2012, Article ID 856523, 17 pages, 2012.
- [3] S. Y. Chen and Y. F. Li, "Vision sensor planning for 3-D model acquisition," *IEEE Transactions on Systems, Man, and Cybernetics B*, vol. 35, no. 5, pp. 894–904, 2005.
- [4] C. Carlo, C. Shengyong, and A. Gani, "Information and modeling in complexity," *Mathematical Problems in Engineering*, vol. 2012, Article ID 868413, 4 pages, 2012.
- [5] S. Y. Chen, Y. F. Li, Q. Guan, and G. Xiao, "Real-time three-dimensional surface measurement by color encoded light projection," *Applied Physics Letters*, vol. 89, no. 11, Article ID 111108, 2006.
- [6] T. Weise, B. Leibe, and L. Van Gool, "Accurate and robust registration for in-hand modeling," in *Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition (CVPR '08)*, June 2008.
- [7] T. Weise, S. Bouaziz, and H. Li, "Realtime performance-based facial animation," in *Proceedings of the 38th Special Interest Group on Computer Graphics and Interactive Techniques (SIGGRAPH '11)*, Vancouver, Canada, August 2011.
- [8] F. Huber and M. Hebert, "Fully automatic registration of multiple 3D data sets," in *Proceedings of the IEEE Workshop on Computer Vision Beyond the Visible Spectrum: Methods and Applications (CVBVS '01)*, Kauai, Hawaii, USA, December 2001.
- [9] T. Jaeggli, T. Koninckx, and L. V. Gool, "Online 3d acquisition and model integration," in *Proceedings of IEEE International Workshop on Projector-Camera Systems (PROCAMS '03)*, Nice, France, 2003.
- [10] S. Azernikov and A. Fischer, "A new volume warping method for surface reconstruction," *Virtual and Physical Prototyping*, vol. 1, no. 2, pp. 65–71, 2006.
- [11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli et al., "Kinectfusion: real-time 3D reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*, pp. 559–568, ACM, New York, NY, USA, 2011.
- [12] R. A. Newcombe, S. Izadi, O. Hilliges et al., *Kinect Fusion: Real-Time Dense Surface Mapping and Tracking*, IEEE ISMAR, 2011.
- [13] L. Yong-Wan, L. Hyuk-Zae, Y. Na-Eun et al., "3-D reconstruction using the kinect sensor and its application to a visualization system," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC '12)*, pp. 3361–3366, 2012.
- [14] M. Zollhöfer, M. Martinek, G. Greiner, M. Stamminger, and J. Süßmuth, "Automatic reconstruction of personalized avatars from 3D face scans," *Computer Animation and Virtual Worlds*, vol. 22, no. 2-3, pp. 195–202, 2011.
- [15] R. Lienhart, A. Kuranov, and V. Pisarevsky, "Empirical analysis of detection cascades of boosted classifiers for rapid object detection," Tech. Rep., Microprocessor Research Lab, 2002.
- [16] K. Low, "Linear least-squares optimization for point-to-plane ICP surface registration," Tech. Rep. TR04-004, University of North Carolina, 2004.
- [17] B. Curless and M. Levoy, "Volumetric method for building complex models from range images," in *Proceedings of the 1996 Special Interest Group on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*, pp. 303–312, August 1996.
- [18] S. D. Roth, "Ray casting for modeling solids," *Computer Graphics and Image Processing*, vol. 18, no. 2, pp. 109–144, 1982.

- [19] W. E. Lorensen and H. E. Cline, "Marching cubes: a high resolution 3D surface construction algorithm," *ACM Transactions on Graphics*, vol. 21, no. 4, pp. 163–169, 1987.
- [20] T. S. Newman and H. Yi, "A survey of the marching cubes algorithm," *Computers and Graphics*, vol. 30, no. 5, pp. 854–879, 2006.

