A SHAPE FROM SHADING BASED 3D RECONSTRUCTION USING PROJECTED POLYGON REPRESENTATION NEURAL NETWORK

Mohamad Ivan Fanany *†, Benyamin Kusumoputro*, Itsuo Kumazawa†

* Faculty of Computer Science, University of Indonesia
Kampus Baru UI, Depok, Indonesia
email : kusumo@cs.ui.ac.id
† Dept. of Computer Science, Graduate School of Information Science and Engineering
Tokyo Institute of Technology, Japan
West 8E building, Ookayama Meguro-ku, Tokyo 227-0052
email : fanany@kmi.cs.titech.ac.jp

ABSTRACT

In this paper, we present a 3D shape modeling system based on Tsai-Shah shape from shading (SFS) algorithm. This SFS provides partial 3D shapes as depth maps of the object to be reconstructed. Our previously developed Projected Polygon Representation Neural Network (PPRNN) performed the reconstruction process. This neural network is able to successively refine the polygon vertices parameter of an initial 3D shape based on 2D images taken from multiple views. The reconstruction is finalized by mapping the texture of object image to the 3D initial shape. It is known from static stereo analysis that even though multiple view images are used, obtaining 3D structure without considering of base-distance information, i.e., focal separation between different camera positions, is impossible. Unless there is something else is known about the scene. Here we propose the use of shading features to extract the 3D depth maps by using a fast SFS algorithm, instead of rendering the object based on bare 2D images. A beginning result of reconstructing human (mannequin) head and face is presented. From our experiment, it was shown that using only 2D images would result a poor reconstruction. While using the depth-maps provides a smoother and more realistic 3D object.

Keywords : Shape from Shading, 3D reconstruction, neural network

1. INTRODUCTION

The shape reconstruction of objects is very complex. The base for the development of such a system is always the human eye. Since our ability to see three-dimensional structures are based on different cues or features of the human eye, e.g., shading, stereo, texture, motion, etc., we can try to emulate one or some of these features. There were several approaches, which reconstructed a single depth-map using one of those features or combination of them [1-3]. However, still in general the reconsiruction of thorough 3D object from 2D images is very difficult subject. So far, a satisfying result of reconstructing a complex object, such as human head and face, which able to maintain an adequate detail, could only be performed with the availability of several hundred of images obtained from an expensive laser scan [4].

In this paper, we took the shading feature of several 2D images to provide partial depth maps of the 3D object for reconstruction of human head and face using PPRNN [5]. The choice of reconstructing human head and face is driven much by current wide application demands in many fields such as animation, human computer interface, automatic identification, and medical survey. However, reconstruction of human head shape, which could maintain its identity, is extremely difficult. This is because human head shape is amazingly consistent across billions of people. Hence gross structure is invariably the same. The variability from one head to another is, in fact, relatively small. Nevertheless, it is these small deviations (on average on the order of a centimeter) that give a face its unique identity.

The presentation of this paper is organized as follows. First, the SFS algorithm used in this paper will be presented in the next section. In the Section 3, the architure and learning mechanism of our PPRNN is described. Camera calibration method using epipolar geometry, is given in the Section 4. Finally, the result of experiment and conclussion are illustrated respectively in Section 5 and 6.

Accepted paper [16 October 2001]. Last Revision [25 November 2001]

2. TSAI-SHAH SFS

Algorithms for the recovery of shape from shading have been investigated extensively, where the performance of some representative algorithms are compared and analyzed [6]. It was reported that generally, the global minimization techniques yields a very good results, while the local approaches tend to have more error but a lot faster. In this report, we used the Tsai-Shah SFS algorithm [7], which is a very fast technique belong to the local approach, assumes a linearity of reflectance map in term of depth.

Tsai-Shah employed the discrete approximations of p and q, using finite differences in order to linearize the reflectance map in terms of depth Z. The reflectance function for Lambertian surfaces is:

$$R(p_{i,j}, q_{i,j}) = \frac{-s_x p_{i,j} - s_y q_{i,j} + s_z}{\sqrt{1 + p_{i,j}^2 + q_{i,j}^2}}. \quad (1)$$

Using the following discrete approximations for p and q, $p = Z_{i,j} - Z_{i-1,j}$ and $p = Z_{i,j} - Z_{i,j-1}$, the reflectance equation can be rewritten as:

$$\begin{aligned} 0 &= f(I_{i,j}, Z_{i,j}, Z_{i-1,j}, Z_{i,j-1}) \\ &= I_{i,j} - R(Z_{i,j} - Z_{i-1,j}, Z_{i,j} - Z_{i,j-1}) \end{aligned} \quad (2)$$

For a fixed point (i, j) and a given image I, a linear approximation (Taylor series expansion up through the first order terms) of the function f (2) about a given depth map Z^{n-1} is:

$$\begin{aligned} 0 &= f(I_{i,j}, Z_{i,j}, Z_{i-1,j}, Z_{i,j-1}) \\ &\approx f(I_{i,j}, Z_{i,j}^{n-1}, Z_{i-1,j}^{n-1}, Z_{i,j-1}^{n-1}) + (Z_{i,j} - Z_{i,j}^{n-1}) \\ &\frac{\partial}{\partial Z_{i,j}} f(I_{i,j}, Z_{i,j}^{n-1}, Z_{i-1,j}^{n-1}, Z_{i,j-1}^{n-1}) + (Z_{i-1,j} - Z_{i-1,j}^{n-1}) \\ &\frac{\partial}{\partial Z_{i-1,j}} f(I_{i,j}, Z_{i,j}^{n-1}, Z_{i-1,j}^{n-1}, Z_{i,j-1}^{n-1}) + (Z_{i,j-1} - Z_{i,j-1}^{n-1}) \\ &\frac{\partial}{\partial Z_{i,j-1}} f(I_{i,j}, Z_{i,j}^{n-1}, Z_{i-1,j}^{n-1}, Z_{i,j-1}^{n-1}). \end{aligned} \quad (3)$$

For an M by N image, there are M^2 such equations, which will form a linear system. This system can be solved easily using the Jacobi iterative scheme, which simplifies equation (3) into the following equation (4):

$$0 = f(Z_{i,j}) \approx f(Z_{i,j}^{n-1}) + (Z_{i,j} - Z_{i,j}^{n-1}) \frac{d}{dZ_{i,j}} f(Z_{i,j}^{n-1}). \quad (4)$$

Then, for $Z_{i,j} = Z_{i,j}^{n}$, the depth map at the n-th iteration, can be solved directly:

$$Z_{i,j}^{n} = Z_{i,j}^{n-1} + \frac{-f(Z_{i,j}^{n-1})}{\frac{d}{dZ_{i,j}} f(Z_{i,j}^{n-1})} \quad (5)$$

The initial estimate of $Z_{i,j}^{0}$ is set to zero for all pixels, Gaussian smoothing is applied to the final depth map to get a smoother result. Note that depth is computed by using a simple division without any matrix inversion (5). This is a simple but efficient algorithm. It was reported that the result of this SFS algorithm works well on smooth lambertian objects with the light source close to the viewing direction. However, it has problems with self-shadows and special care is needed to avoid division by zero.

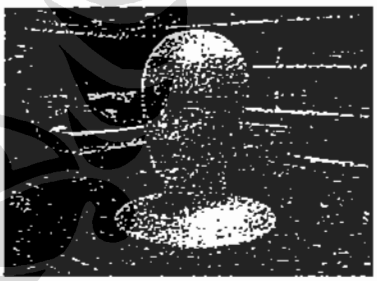

Figure 1. The object to be reconstructed

In this report, the object to be reconstructed is shown in Fig. 1. It is a replica of a smooth human (mannequin) head, assumed to have constant albedo. The object is viewed from three different views with slant angles 0, -45, +45 degrees, with the source location (0.01, 0.01, 1). The images and its corresponding depth-maps (partial shapes) are shown in Fig. 2. It could be observed that a reasonable result was obtained, except for several regions around eyes and nose, which contain self-shadows.

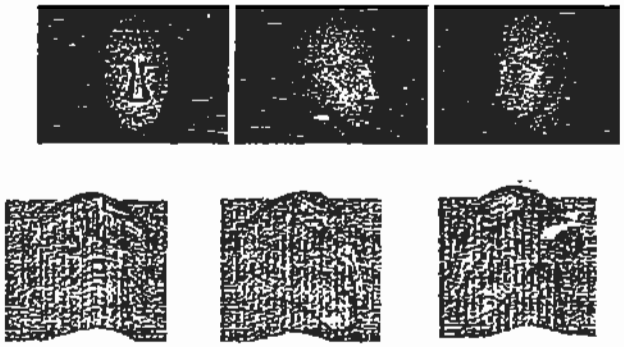

Figure 2. Three different views and its depth maps

3. PROJECTED POLYGON REP. NN

Recently, we developed a 3D shape modeling system using the shape's multiple views to determine the polygon parameters [8]. This newly designed neural network aims to reconstruct 3D object based on images taken from multiple views, without the need to know the base-distance and focal distance. We named this neural network setting as PPRNN. The scheme of the use of PPRNN for our 3D reconstruction system is depicted in Fig. 3.

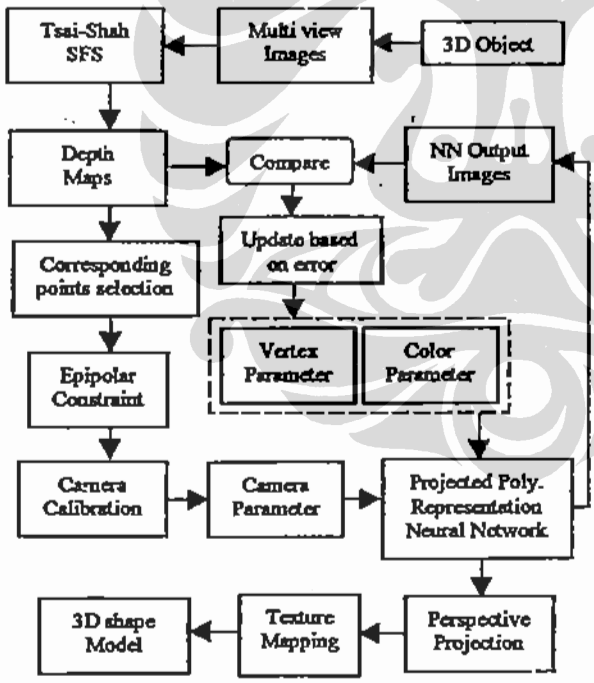

Figure 3. The PPRNN learning scheme

Using this neural network, each of which represents a view, we could determine the polygon parameters by error back propagation algorithm. The advantage of applying this neural network approach is that it provides a fast and integrated system for modeling 3D object from an initial deformable 3D shape. However, it was shown that the generation of the 3D model based on 2D images (image-based rendering) using this PPRNN, is hard to converge especially for complex object, and could only result two-dimensional structure approximation. In this report, we extend the generation performance of this neural network, not based on 2D images but 3D depth-maps obtained from Tsai-Shah SFS algorithm.

3.1. PPRNN ARCHITECTURE

In this system, we generate a polygon model of an object in the three-dimensional space. As shown in Fig. 4, a 3D object in three-dimensional space is projected to 2D screen using perspective projection. Where R, T, A are the camera parameters. R and T are rotation and translation matrices to transform the vertices of an object according to posing direction and position of the camera. A is a transformation matrix which comprises the focus distance, skewing, and scaling of the camera. Therefore, R and T are called as extrinsic camera parameters, while A is called as intrinsic camera parameters.

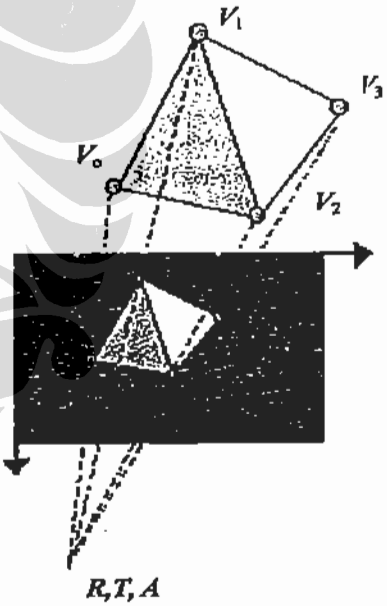

Figure 4. Polygon model projection from 3D to 2D

Our PPRNN is able to represents 2D shape of polygonal object as its output. A polygon is formed by triangles. A triangle is formed by three straight-line edges. Following this way, PRNN (Polygon

Representation Neural Network) could be constructed by combining a number of TRNNs (Triangle Representation Neural Network) for triangles representation, while each of TRNN consisted of three ERNNs (Edge Representation Neural Network) for straight-line edges representation forming the triangle. This neural network structure is shown in Fig. 5-7. By attaching the *Project Unit* to the PRNN, we could project the 3D vertices into 2D, using perspective projection, hence forming the PPRNN. Based on the error between the depth-map of output images of PPRNN and the depth-map of target images, the vertex parameters of 3D vertices of PPRNN are adjusted

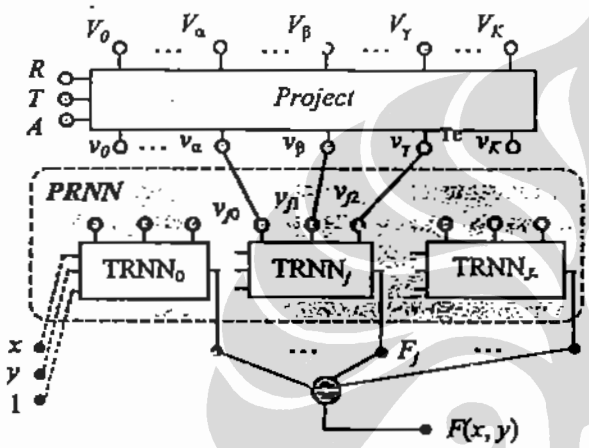

Figure 5. The PPRNN structure

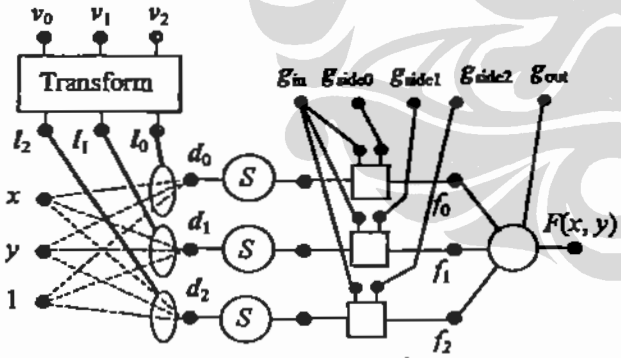

Figure 6. The TRNN structure

Basic operation of this PPRNN could be explained as follows. For an input pixel location (x, y) we have to determine the output $F(x, y)$. This output value is actually a pixel value of the output image (projection image) of PPRNN. In determining this pixel value, we have to find out whether this pixel is 'inside' or 'outside' projected polygon area. If it is outside, we define its output to g_{out}. If it is inside, then we have to find out, to which triangle it is belong. After this triangle is found, we have to determine, to which side it is belong. Then the pixel value is assigned to g_{in}. While for every triangle, the pixel values inside are set to g_{in}, the pixel values outside are set to g_{side} $fj = 0,1,2$) since every triangle adjacently has three neighbor. Special care has to be performed in case of a triangle is around the edge of polygon. There are some switching mechanism in selecting those triangles and edges. For further explanation, could be referred to [5]. It is important to be noted here that, since we use the depth-maps instead of 2D images, the term images above, should be replaced by depth-maps.

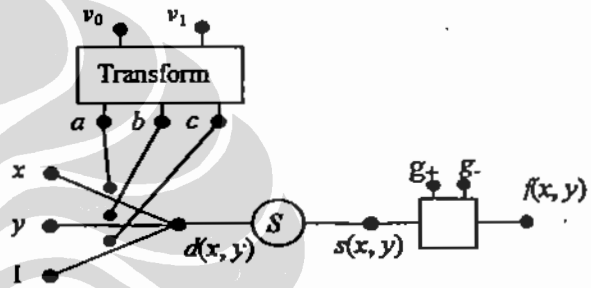

Figure 7. The ERNN structure

It is essential to understand the structure details of ERNN. By understanding this most primitive structur, we could figure out the working mechanism of the overall system, i.e., TRNN, PRNN, PPRNN, more easily.

Actually, the ERNN used sigmoid function to represent straight line edge. The used of sigmoid function is suggested by the necesity to have a continuous (differentiable) function for neural network learning.

If a line l: $ax + by + c = 0$ is defined, we could state *correct-area* and *false-area* as follows,

$$\text{correct-area} \quad : ax + by + c \geq 0 \qquad (6)$$

$$\text{false-area} \quad : ax + by + c < 0 \qquad (7)$$

Through the straigth-line parameter $[a, b, c]^T$, we could define its orientation.

As shown in Fig. 8, consider the representation of edge image formed by a straight-line

$\hat{l}: \hat{a}x + \hat{b}y + \hat{c} = 0$ which pass through two points $\hat{v}_0 = [\hat{x}_0, \hat{y}_0]^T$ and $\hat{v}_1 = [\hat{x}_1, \hat{y}_1]^T$. The color of correct-area is $\hat{g}_+ = [\hat{r}_+, \hat{g}_+, \hat{b}_+]^T$, and for false-area is $\hat{g}_- = [\hat{r}_-, \hat{g}_-, \hat{b}_-]^T$. In the case that the 2 points are defined, the orientation of the straight-line could be stated.

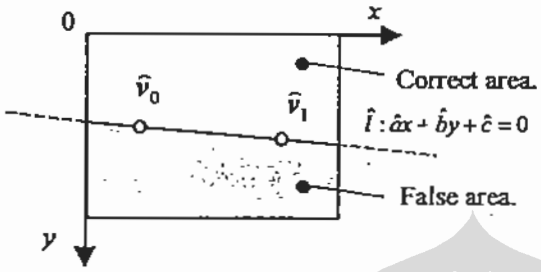

Figure 8. Straight line edge formed by ERNN

$[x, y]^T$ is the input part, while $f(x, y)$ is the output part. $v_0 = [x_0, y_0]^T$, $v_1 = [x_1, y_1]^T$ are called as point parameters. A line l is created from this 2 points. $g_+ = [r_+, g_+, b_+]^T$, $g_- = [r_-, g_-, b_-]^T$ are called as color parameters. The input of ERNN is image's coordinate value $[x, y]^T$, and the output is a pixel value $f(x, y) = [r, g, b]^T$ at the corresponding coordinate that is defined in equations below.

$$l' = \begin{pmatrix} a' \\ b' \\ c' \end{pmatrix} = \begin{pmatrix} y_1 - y_0 \\ x_1 - x_0 \\ x_1 y_0 - x_0 y_1 \end{pmatrix} \quad (8)$$

$$l = \frac{1}{\sqrt{a'^2 + b'^2}} l' \quad (9)$$

$$\alpha = g_+ - g_- \quad (10)$$

$$\beta = g_- \quad (11)$$

$$d(x, y) = ax + by + c \quad (12)$$

$$s(x, y) = sigmoid(d(x, y), \alpha) \quad (13)$$

$$f(x, y) = s(x, y)\alpha + \beta \quad (14)$$

The Transform-Unit is part which transform the 2 points v_0, v_1 to straight-line parameter $l = [a, b, c]^T$. Since the straight-line parameter l is normalized such as $a^2 + b^2 = 1$, $d(x, y)$ of the equation (12) state the distance between point (x, y) and straight-line l. The output is near g_+ value at correct-area, and near g_- value at the false-area.

The point parameter v_0, v_1 are the 2 arbitrary different points on the straight line of the image that would be represented by ERNN. The color parameter g_+, g_- is the color of false area and the color of correct-area of the image that would be represented. When we want to represent the image as shown in Fig. 8, all of $\hat{v}_0, \hat{v}_1, \hat{g}_+, \hat{g}_-$ have to be defined. By setting up the parameter v_0, v_1, g_+, g_- of ERNN into various values, any straight-line edge could be represented.

3.2. PPRNN LEARNING ALGORITHM

The camera parameters of n-th image $G^{(n)}$, i.e., $R^{(n)}$, $T^{(n)}$, $A^{(n)}$, are computed from correspondence points with other images. This computation method using epipolar constraint will be presented in Section 4. After the camera parameters: $R^{(n)}$, $T^{(n)}$, $A^{(n)}$, are set, the neural network will throw out the value of its output $F^{(n)}$. Then, by comparing between projection images and target images taken from several different views, vertices V_k could be learned by using PPRNN.

The color parameters g_j are learned as follows. The projection region of triangle T_j is $P_j (\subset R^2)$ $(j = 0,1,2\ldots)$. The region, which is not included in one of triangle projection region, is set as $P_{out} (\subset R^2)$. In each step of learning, g_j is revised to the average pixel value of region P_j in the teacher image.

The vertex parameters V_k are learned using error back propagation rule. After appropriate initial values are given, the error evaluation function E is defined by measuring the difference between depth value in the real (teacher) image $G^{(n)}(x, y)$ with the depth value of neural network output $F^{(n)}(x, y)$. Until this error E is become small enough, V_k are learned. This learning is performed by iteratively compute the gradient descent following equation below:

$$V_k = V_k - \eta \frac{\partial E}{\partial V_k} \qquad (k = 1,2,\cdots,K) \quad (15)$$

where η learning rate constant, a small positive real value. The evaluation function E is defined as follows:

$$e^{(n)}(x, y) = \left| F^{(n)}(x, y) - G^{(n)}(x, y) \right|^2 \quad (16)$$

$$\varepsilon^{(n)} = \frac{1}{S^{(n)}} \sum_{x,y} e^{(n)}(x, y) \quad (17)$$

$$E = \frac{1}{N}\sum_{n} \varepsilon^{(n)} \tag{18}$$

where $S^{(n)}$ is the number of pixel in n-th image. N is the number of image used for learning.

$e^{(n)}(x, y)$ is the mean squared error between n-th depth map of neural network $F^{(n)}$ and n-th depth map of teacher image $G^{(n)}$ at location (x, y). The average of this error over all the images is the evaluation function E.

From (17)(18), we could determine

$$\frac{\partial E}{\partial V_k} = \frac{1}{N}\sum_{n}\frac{1}{S^{(n)}}\sum_{x,y}\frac{\partial e^{(n)}(x,y)}{\partial V_k}. \tag{19}$$

If $\partial e^{(n)}(x,y)/\partial V_k$ could be found, the vertex V_k in (15) could be learned. $\partial e^{(n)}(x,y)/\partial V_k$ is back propagated signal determined using back propagation learning rule, input in backward direction to the output $F^{(n)}$ of neural network. Hence the backward signal is $\partial e^{(n)}(x,y)/\partial F^{(n)}(x,y)$.

$\partial e/\partial v_k$ could be found by observing the propagation of backward signal using error backpropagation method after the coordinate value $[\, x, y \,]^T$ is inputted to the ERNN. When Δf is inputted to the output section in backward direction, this neural network will operate such as

$$\Delta v_k = \frac{\partial f^T}{\partial v_k}\Delta f \qquad (k = 0, 1) \tag{20}$$

is outputted to the part of parameter v_k ($k = 0, 1$). Therefore, if the input of backward flow is $\Delta f = \partial e/\partial f$, and the output at part of parameter v_k ($k = 0, 1$) become

$$\Delta v_k = \frac{\partial f^T}{\partial v_k}\Delta f \tag{21}$$

$$= \frac{\partial f^T}{\partial v_k}\frac{\partial e}{\partial f} \tag{22}$$

$$= \frac{\partial e}{\partial v_k} \qquad (k = 0, 1) \tag{23}$$

Here, in backward stage ERNN operates as equation (7.30), each unit is passed by backward signal, presented by equations below.

$$\Delta s = \frac{\partial f^T}{\partial s}\Delta f \tag{24}$$

$$= \alpha^T \Delta f \tag{25}$$

$$= (g_+ - g_-)^T \Delta f \tag{26}$$

$$\Delta d = \frac{\partial s}{\partial d}\Delta s \tag{27}$$

$$= \alpha d(1-d)\Delta s \tag{28}$$

$$\Delta l = \frac{\partial d^T}{\partial l}\Delta d \tag{29}$$

$$= \tilde{x}^T \Delta d \tag{30}$$

$$\Delta l' = \frac{\partial l^T}{\partial l}\Delta l \tag{31}$$

$$= \left(\frac{1}{\sqrt{a'^2 + b'^2}} I - \frac{1}{\left(\sqrt{a'^2 + b'^2}\right)^3} \begin{bmatrix} a' \\ b' \\ 0 \end{bmatrix} \right)^T \Delta l \tag{32}$$

$$\Delta v_0 = \frac{\partial l'^T}{\partial v_0}\Delta l' \tag{33}$$

$$= \begin{bmatrix} 0 & 1 & y_1 \\ -1 & 0 & x_1 \end{bmatrix} \Delta l' \tag{34}$$

$$\Delta v_1 = \frac{\partial l'^T}{\partial v_1}\Delta l' \tag{35}$$

$$= \begin{bmatrix} 0 & 1 & y_0 \\ -1 & 0 & x_0 \end{bmatrix} \Delta l' \tag{36}$$

provided that, $\tilde{x} = [x, y, 1]^T$. The backward input is

$$\Delta f = \frac{\partial e}{\partial f} \tag{37}$$

$$= 2(f - G) \tag{38}$$

After the above is proceed, we perform the following learning algorithm.

Learning algorithm.
Until the evaluation function is small enough, the following steps are performed iteratively.

1. The pixel values of each area P_+, P_- on the training image are averaged, hence g_+, g_- the are updated.
2. $\partial e/\partial v_k$ ($k = 0, 1$) of all pixels are found using backpropagation method. The coordinate $[x, y]^T$ is inputted to the neural network. After the output f is obtained, if $2(f - G)$ is inputted in backward direction, $\partial e/\partial v_k$ is outputted from the part of vertex parameter v_k.
3. From the next equation, $\partial e/\partial v_k$ ($k = 0, 1$) is found.
4. $$\frac{\partial \varepsilon}{\partial v_k} = \frac{1}{S}\sum_{x,y}\frac{\partial e}{\partial v_k} \qquad (k = 0, 1) \quad (39)$$
5. From the next equation, the 2 points parameter which lied on a straight-line is updated.
6. $$v_k = v_k - \eta\frac{\partial \varepsilon}{\partial v_k} \qquad (k = 0, 1) \quad (40)$$

So far, we show only the essentill part of the PPRNN learning mechanism, that is the ERNN learning. For the TRNN, PRNN, and PPRNN, there is some mechanisme to switch and connect the g_+ and g_- of the ERNN to the g_{side0}, g_{side1}, g_{side2}, g_{in} and g_{out} of TRNN. And finally the mechanism to switch and connect the output of TRNN to the output of PRNN and PPRNN, that is its g_{in} and g_{out}.

4. CAMERA CALIBRATION

In this system, before we could use the PPRNN, the camera parameters should be set at first. Camera parameters are computed, by identifying geometrical relation between corresponding points of multi view images. Computation method will be explained below.

4.2. CAMERA PARAMETER

R is three-dimensional rotation matrix represent s camera's posing direction; T three-dimensional vector represents camera position.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \quad (41)$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad (42)$$

$$R_z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (43)$$

$$R = R_x R_y R_z \quad (44)$$

$$T = (t_x, t_y, t_z)^T \quad (45)$$

A is 3x3 matrix for transforming the physical coordinate (x, y) into computer image coordinate (u, v) defined as:

$$A = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (46)$$

(u_0, v_0) is the center of image, while are defined by using focal distance f, scaling factor $k_u k_v$, and skewing constant k_s, as

$$(\alpha_u, \alpha_v, s)^T = f(k_u, k_v, k_s)^T \quad (47)$$

4.2. PERPECTIVE PROJECTION

After camera parameters R, T, A are defined, a coordinate transformation of a point in 3D world coordinate $X = (X, Y, Z)$ to camera coordinate $X_c = (X_c, Y_c, Z_c)$ could be performed. This transformation is followed by a perspective transformation into $m = (u, v)$ as stated in (18)-(20).

$$X_c = RX + T \quad (48)$$

$$x = \begin{pmatrix} X_c / Z_c \\ Y_c / Z_c \end{pmatrix} \quad (49)$$

$$\tilde{m} = A\tilde{x} \quad (50)$$

$\tilde{m}, \tilde{x}$ are the augmented (homogenous) vector of m, x i.e., $\tilde{x} = (x, y, 1)^T$ and $\tilde{m} = (u, v, 1)^T$. The *Project* unit in the PPRNN, transform the vertex coordinate V_k on 3D space to its 2D projection image coordinate v_k

4.3. EPIPOLAR GEOMETRY

The calibration of camera parameters is performed by using epiporal geometry. A set of corresponding points from at least two different view images have to be used. First, consider a point in world space to be viewed by camera no. 1. Based on that description then we think how if the same point is viewed by camera no. 2. A point X in 3D space viewed from the coordinate of camera no. 1 view point, is transformed to the coordinate of camera no. 2. This transformation is composed of rotation represented by 3x3 matrix R and 3x1 translation vector T

$$X' = RX + T \tag{51}$$

The intrinsic camera parameter for each of the two cameras are named A and A'.

Point X are projected to two points of both cameras as m and m'. Provided that those projection points are known, the following relation could be stated:

$$\tilde{m}'^T F \tilde{m} = 0 \tag{52}$$

where,

$$F = A'^{-T} E A^{-1} \tag{53}$$

$$E = [T]_\times R \tag{54}$$

when $T = (t_x, t_y, t_z)^T$, the $[T]_\times$ could be defined as

$$[T]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}. \tag{55}$$

Equation (52) is called as *epipolar equation*. E is called as essential matrix and F is called as fundamental matrix. Taking two image from two different views, the correspondence points certainly has to satisfy equation (52). This is called as *epipolar constraint*.

4.4. COMPUTATION METHOD

There are many methods to compute the camera parameter from correspondence points by using epipolar geometry [8].

First, a set of correspondence points from two images (m_i, m_i') $(i=1,2,...,M)$ is prepared. Next, by using the left side of epipolar equation (52), the following evaluation function of camera parameter,

$$C = \frac{1}{M} \sum_i \tilde{m}_i'^T F \tilde{m}_i \tag{56}$$

is declared. After the camera parameters is given an appropriate initial values, as the C become smaller, the camera parameter are iteratively computed by gradient descent rule. After this computation is performed to all combination of images, then the camera parameters of all images are defined.

Here, intrinsic parameters $\alpha_u, \alpha_v, s, u_0, v_0$, and extrinsic parameters $\phi, \theta, \varphi, T_x, T_y, T_z$, are updated as follows. As p represents one of those parameters,

$$p = p - \eta \frac{\partial C}{\partial p} \tag{57}$$

where η is a small constants which represents correction-measure of parameter adjustment. Hence for each parameter is determined as follows.

$$\frac{\partial C}{\partial \alpha_u} = \frac{1}{M} \sum_i \tilde{m}_i'^T \left(\frac{\partial A^{-T}}{\partial \alpha_u} E A^{-1} + A^{-T} E \frac{\partial A^{-1}}{\partial \alpha_u} \right) \tilde{m}_i' \tag{58}$$

$$\frac{\partial C}{\partial \alpha_v} = \frac{1}{M} \sum_i \tilde{m}_i'^T \left(\frac{\partial A^{-T}}{\partial \alpha_v} E A^{-1} + A^{-T} E \frac{\partial A^{-1}}{\partial \alpha_v} \right) \tilde{m}_i' \tag{59}$$

$$\frac{\partial C}{\partial s} = \frac{1}{M} \sum_i \tilde{m}_i'^T \left(\frac{\partial A^{-T}}{\partial s} E A^{-1} + A^{-T} E \frac{\partial A^{-1}}{\partial s} \right) \tilde{m}_i' \tag{60}$$

$$\frac{\partial C}{\partial u_0} = \frac{1}{M} \sum_i \tilde{m}_i'^T \left(\frac{\partial A^{-T}}{\partial u_0} E A^{-1} + A^{-T} E \frac{\partial A^{-1}}{\partial u_0} \right) \tilde{m}_i' \tag{61}$$

$$\frac{\partial C}{\partial v_0} = \frac{1}{M} \sum_i \tilde{m}_i'^T \left(\frac{\partial A^{-T}}{\partial v_0} E A^{-1} + A^{-T} E \frac{\partial A^{-1}}{\partial v_0} \right) \tilde{m}_i' \tag{62}$$

$$\frac{\partial C}{\partial \phi} = \frac{1}{M} \sum_i \tilde{m}_i'^T A^{-T} [T]_\times \frac{\partial R_x}{\partial \phi} R_y R_z A^{-1} \tilde{m}_i \tag{63}$$

$$\frac{\partial C}{\partial \theta} = \frac{1}{M} \sum_i \tilde{m}_i'^T A^{-T} [T]_\times R_x \frac{\partial R_y}{\partial \theta} R_z A^{-1} \tilde{m}_i \tag{64}$$

$$\frac{\partial C}{\partial \varphi} = \frac{1}{M} \sum_i \tilde{m}_i'^T A^{-T} [T]_\times R_x R_y \frac{\partial R_z}{\partial \varphi} A^{-1} \tilde{m}_i \tag{65}$$

$$\frac{\partial C}{\partial T_x} = \frac{1}{M}\sum_i \tilde{m}_i'^T A^{-T} \frac{\partial [T]_\times}{\partial T_x} R A^{-1} \tilde{m}_i \quad (66)$$

$$\frac{\partial C}{\partial Ty} = \frac{1}{M}\sum_i \tilde{m}_i'^T A^{-T} \frac{\partial [T]_\times}{\partial T_y} R A^{-1} \tilde{m}_i \quad (67)$$

$$\frac{\partial C}{\partial T_z} = \frac{1}{M}\sum_i \tilde{m}_i'^T A^{-T} \frac{\partial [T]_\times}{\partial T_z} R A^{-1} \tilde{m}_i \quad (68)$$

provided that,

$$\frac{\partial A^{-1}}{\partial \alpha_u} = \begin{bmatrix} -\frac{1}{\alpha_u^2} & -\frac{s}{\alpha_u^2 \alpha_v} & \frac{\alpha_u \alpha_v - s v_0}{\alpha_u^2 \alpha_v} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (69)$$

$$\frac{\partial A^{-1}}{\partial \alpha_v} = \begin{bmatrix} 0 & -\frac{s}{\alpha_u \alpha_v^2} & -\frac{s v_0}{\alpha_u \alpha_v^2} \\ 0 & -\frac{1}{\alpha_v^2} & -\frac{v_0}{\alpha_v^2} \\ 0 & 0 & 0 \end{bmatrix} \quad (70)$$

$$\frac{\partial A^{-1}}{\partial s} = \begin{bmatrix} 0 & \frac{1}{\alpha_u \alpha_v} & -\frac{v_0}{\alpha_u \alpha_v} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (71)$$

$$\frac{\partial A^{-1}}{\partial u_0} = \begin{bmatrix} 0 & 0 & -\frac{1}{\alpha_u} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (72)$$

$$\frac{\partial A^{-1}}{\partial v_0} = \begin{bmatrix} 0 & 0 & \frac{s}{\alpha_u \alpha_v} \\ 0 & 0 & -\frac{1}{\alpha_v} \\ 0 & 0 & 0 \end{bmatrix} \quad (73)$$

$$\frac{\partial R_x}{\partial \phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\sin\phi & -\cos\phi \\ 0 & \cos\phi & -\sin\phi \end{bmatrix} \quad (74)$$

$$\frac{\partial R_y}{\partial \theta} = \begin{bmatrix} -\sin\theta & 0 & \cos\theta \\ 0 & 1 & 0 \\ -\cos\theta & 0 & -\sin\theta \end{bmatrix} \quad (75)$$

$$\frac{\partial R_z}{\partial \varphi} = \begin{bmatrix} -\sin\varphi & -\cos\varphi & 0 \\ \cos\varphi & -\sin\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (76)$$

$$\frac{\partial [T]_\times}{\partial T_x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad (77)$$

$$\frac{\partial [T]_\times}{\partial T_y} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad (78)$$

$$\frac{\partial [T]_\times}{\partial T_z} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (79)$$

The evaluation equation of m-th image and n-image (56), could be notated as $C_{m,n}$. When the total number of image become N, the average of $C_{m,n}$ of all images are defined as

$$\overline{C} = \frac{1}{{}_N C_2}\sum_{m \neq n} C_{m,n} \quad (80)$$

where ${}_N C_2$ is the total number of combination when two images are chosen from N images.Until this $\overline{C}$ is small enough, each camera parameter is updated according to equation (57).

5. RESULT OF EXPERIMENT

Here, we conducted an experiment to compare 3D reconstructed shape with SFS and without SFS after 1000 iterations. Images from reconstructed 3D shape, viewed from several postures are presented in Fig. 9.

Compared to the result without using SFS algorithm, using SFS (Tsai-Shah approach) is experimentally shown as faster to converge resulting a smoother and more realistic reconstructed 3D object.

As for rough approximation to the shape of 3D object, the reconstruction of PPRNN based on Tsai-Shah SFS algorithm is capable to provide a reasonable approximation. However, for a detail, it was still a poor result. This might be caused by some problems. Firstly, there might be un-uniform light source condition, since so far we controlled this light-source manually. Secondly, there are some parts of the object which is obstructed by shadow, for example the area around eyes and nose.

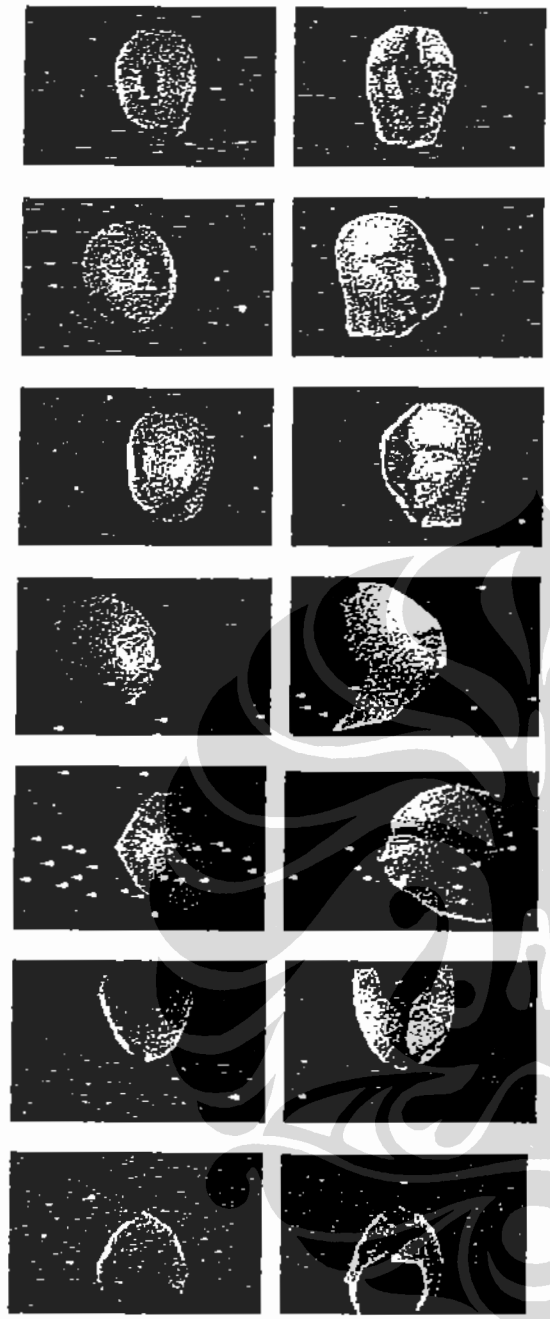

Figure 9. Result of 3D reconstruction with SFS (left mages), and without SFS (right images)

5. CONCLUSION

In this paper, we proposed the use of SFS to provide partial 3D shapes for reconstruction 3D object using PPRNN. We observed several issues and directions for our further study.

First, one important shortcoming of PPRNN is that it generates only 'flat shading' in its output while learning. This is due to inevitable used of sigmoid function which causing the pixel value to be flat after its saturation curve. Currently, we are trying to resolve this, so that it could generate 'smooth shading' (in this case we used Gouraud shading) which interpolated the pixel intensity biliearly, based on the pixel intensity of the vertices of polygon.

Secondly, a quantitative measurement of real 3D object, e.g., using 3D scanner, need to be performed, in order to measure the performance quantitatively. Thirdly, The Tsai-Shah SFS algorithm, which is belong to linear approximation class should be compared to other SFS methods. Finally, in order to obtain more accurate result the SFS algorithms should be combined with other features, e.g., stereo, motion and the very important one: shadow.

REFERENCES

[1] O.E. Vega and Y.H. Yang, "*Shading Logic: A Heuristic Approach to Recover Shape From Shading*, " IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 15, no. 6, pp. 592-597, June 1993.

[2] A. Pentland, "*Photometric Motion*," IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 13, pp. 879-890, 1991.

[3] J.Cryer, P.Tsai, M. Shah, "*Integration of Shape from x Modules: Combining Stereo and Shading*," IEEE Proc. Computer Vision and Pattern Recognition, pp. 720-721, June 1993.

[4] J.J. Atick, P.A. Griffin, A. N. Redlich, "*Statistical Approach to Shape From Shading: Reconstruction of Three Dimensional Face Surfaces from Single Two-Dimensional Images*," Neural Computation, vol. 8, No. 6, pp. 1321-1340, August 1996.

[5] M. Ohno, I. Kumazawa, "*3D shape representation by a neural network and reconstruction of a 3D shape from its multiple views*," Tech. Report of IEICE, pp.23-30, PRMU2000-89

[6] R. Zhang, P. Tsai, J. E. Cryer, M. Shah, "*Shape from Shading: A Survey*," IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 21, No. 8, August 1999.

[7] P. Tsai, M. Shah, "*Shape from Shading Using Linear Approximation*, " Image and Vision Computing J., vol. 12, no. 8, pp. 487-498, 1994.

[8] Z. Zhang, "*Determining the epipolar geometry and its uncertainty: A review*," International Journal of Computer Vision, 1998.