Novel methods for real-time 3D facial recognition

RODRIGUES, Marcos <http://orcid.org/0000-0002-6083-1303> and ROBINSON, Alan

Novel Methods for Real-Time 3D Facial Recognition

Marcos A Rodrigues and Alan Robinson
Geometric Modelling and Pattern Recognition Research Group – GMPR
Computing and Communications Research Centre
Sheffield Hallam University, Sheffield UK
{*m.rodrigues, a.robinson*}@shu.ac.uk
www.shu.ac.uk/gmpr

Abstract

[1]In this paper we discuss our approach to real-time 3D face recognition. We argue the need for real time operation in a realistic scenario and highlight the required pre- and post-processing operations for effective 3D facial recognition. We focus attention to some operations including face and eye detection, and fast post-processing operations such as hole filling, mesh smoothing and noise removal. We consider strategies for hole filling such as bilinear and polynomial interpolation and Laplace and conclude that bilinear interpolation is preferred. Gaussian and moving average smoothing strategies are compared and it is shown that moving average can have the edge over Gaussian smoothing. The regions around the eyes normally carry a considerable amount of noise and strategies for replacing the eyeball with a spherical surface and the use of an elliptical mask in conjunction with hole filling are compared. Results show that the elliptical mask with hole filling works well on face models and it is simpler to implement. Finally performance issues are considered and the system has demonstrated to be able to perform real-time 3D face recognition in just over 1s 200ms per face model for a small database.

1. Introduction

It is often stated that 3D facial recognition offers potential advantages over 2D methods (Cook *et al*., 2006; Bowyer *et al*., 2004) as a number of limitations can be overcome including lighting variations and viewpoint dependency (Medioni and Waupotitsch, 2003; Gordon, 1992). Moreover, the availability of 3D data can provide high accuracy on describing surface features such as cheeks and nose through curvature descriptors (Hesher *et al*., 2003). In the past, we have tested facial recognition algorithms based on geometry alone and have achieved recognition rates up to 97\% (Rodrigues et al., 2008; Brink, 2008). In those experiments, a person was enrolled only once in the database from a standard frontal view and the verification models were reconstructed from non-standard poses with the person facing slightly left or right – a situation that would not be possible with 2D recognition methods with this level of accuracy. In this paper we describe methods that have bettered those results achieving 100% accuracy for a small database of high-density meshes.

A difficulty with 3D face recognition research is the availability of 3D models and the format in which they are presented. The Face Recognition Grand Challenge (FRGC,

[1] In Press by ATINER, Greece. Paper presented at the 5th International Conference on Computer Science and Information Systems 27-30 July 2009, Athens, Greece

2005) has allowed the wider research community to test face recognition algorithms from standard 2D and 3D databases. Concerning 3D facial recognition, one severe limitation of the FRGC is that it was not designed to operate in real-time. 3D data were standardized such that an application would load the pre-formatted data into memory for feature extraction and recognition. 3D data were reconstructed from human subjects taken from a frontal, but arbitrary view point and, given that these are large files containing the structure of the vertices in 3D, this rules out the possibility of testing algorithms in a real-time scenario. Therefore, while 3D data were profitably used to test recognition algorithms in the FRGC, the process does not represent a natural way in which 3D facial recognition systems are to be deployed.

This paper argues the need for real-time 3D facial recognition in a realistic scenario. A major pillar of our argument is that we must control the 3D acquisition process and this process must be extremely fast. If we do not generate our own 3D data then we must rely on loading from standard file format representations such as Wavefront's OBJ, which means that we cannot operate in real-time. To enable real-time operation, this paper puts forward the following *modus operandi* and system requirements: 1) Automatic routines for tracking the face and eyes would operate on the face from live video stream and, when some conditions are met, the image is converted into a 3D structure. 2) The 3D structure would be subject to a number of post-processing operations for noise and outlier removal, smoothing, mesh repair and so on. 3) 3D features are automatically detected and extracted and converted into a set of unique measurements representing a face model. 4) Such features and their measurements are placed into a database to support both identification and verification. 5) Finally, all of the above operations would ideally be computed within 1—2 seconds to allow realistic, real-time operation.

Within our research group we have developed methods for fast 3D reconstruction using line projection (Robinson *et al*., 2004; Brink *et al*., 2008; Rodrigues *et al*., 2008, 2007; Robinson *et al.*, 2006, 2004). The method is based on projecting a pattern of lines on the target surface and processing the captured 2D image from a single shot into a point cloud of vertices in 3D space. Once a point cloud is obtained, it is then triangulated and relevant feature points on the surface model can be detected for recognition. The process is called the 3D Striper Model (3DS) encompassing a number of operations defined as pre- and post-processing:

- Pre-processing operations (2D):
  - face and eye tracking
  - image filtering
  - stripe indexing
- Post-processing operations (3D):
  - generation of 3D point cloud and mesh triangulation
  - noise removal, hole filling, mesh smoothing
  - mesh subdivision
  - pose normalization
  - feature extraction and recognition

This paper focuses on performance issues of face and eye tracking in 2D and noise removal, hole filling, and mesh smoothing in 3D. The objective is to test the

effectiveness of the methods for real time operation and compare the relative merits of 3D post processing concerning various methods such as Laplace and bilinear hole filling, and Gaussian and moving average smoothing. The paper is structured as follows. Section 2 discusses the method and performance of automatic face and eye tracking, Section 3 presents a number of 3D post processing operations and discusses their relative merits. Section 4 presents a method for feature extraction and recognition and finally, a conclusion is presented in Section 5.

2. Automatic Face and Eye Tracking

The Intel's Microcomputer Research Lab has developed a highly optimized computer vision library that is fine-tuned for the underlying processor type. The processor type is automatically detected and the optimized functions can run from 2 to 8 times faster than equivalent optimized C functions (Bradski and Pisarevsky, 2000). Intel libraries come with built-in routines for real-time face detection based on Haar-like features. A great advantage of the Intel libraries is that it is possible to train and use a cascade of boosted classifiers for rapid object detection for any arbitrary object, not only for faces.

There are a number of tutorials available to train OpenCV for rapid object detection such as the one described in (Adolf, 2003; Seo, 2009). Since eye detection is not built-in into OpenCV 1.0 (but it is now included in OpenCV 2.0), we followed the tutorials for training with examples of left and right eye. The general problem with such detection techniques is the number of false positives. For instance, in any image there could be various detected faces and some might not be real faces. We have experienced the same problem with eye detection; the routines normally detect more eyes than there are in the scene.

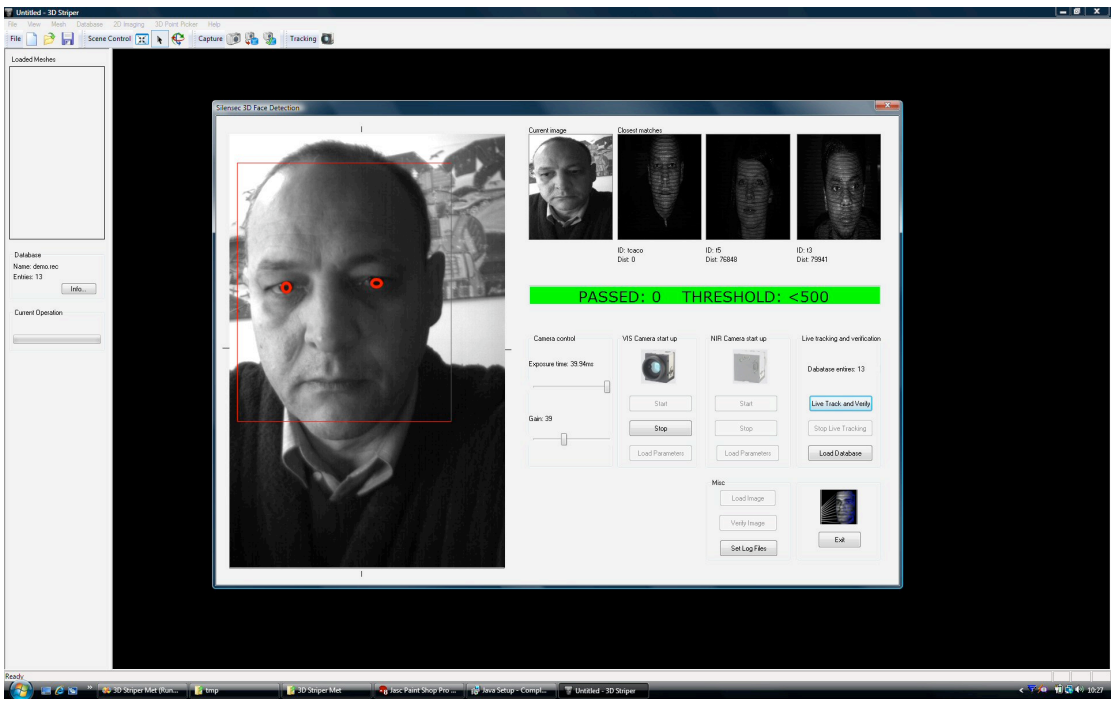

Figure 1: Interface sowing automatic face and eye detection.

In order to deal with false positives and use the libraries in a consistent way, we defined a number of constraints based on the way we wish our system to operate. The face and eye detection run in a separate thread defined within a C++ environment. The problem to address is to decide when to take a shot of the face for 3D reconstruction. Thus, the constraints are simply defined as: first, there should be only one face detected in the image and the face width must be larger than a certain threshold (300 pixels in our case); second, there should be only one left and only one right eye detected in the image, and these must be within the region of interest set by the face detection; third, the position of the face and eyes must not have moved more than a set threshold since last detection (10 pixels in our case) so to avoid taking blurred shots due to undesirable motion.

The system is continuously tracking and detecting (possibly multiple) face and eyes, but only when the above conditions are satisfied a shot is taken (Figure 1). In this way, the system is apparently idle until someone places their face in front of the camera. We have extensively tested Intel's face detection in connection with our own eye detection and it works remarkably well in real-time. It takes only a few moments to lock on the face and eyes, a video demo can be seen in our web pages (GMPR, 2009).

3. 3D Post-Processing

In this section we describe three steps in 3D post-processing namely hole filling, smoothing, and noise removal. The objective is to perform a comparative analysis of the various methods concerning the perceived quality of the resulting model and relative processing time.

Due to the way our 3D acquisition method works (Robinson *et al*., 2004) the 3D structure is a mirror of the stripe pattern that is cast onto the object, where the x dimension represents each stripe index, y represents the index of vertices along a given stripe, and z is the vertex depth. Any vertex $V_{x,y,z}$ in 3D space is defined as a floating point and has an associated Boolean $\boldsymbol{B}_{x,y,z}$ value. If ($\boldsymbol{B}_{x,y,z}$ = TRUE) then $V_{x,y,z}$ is a valid vertex, otherwise it is invalid.

In order to perform operations on the structure, it becomes easy to navigate the data set and test whether a vertex is valid or not. A valid vertex can have its (x,y,z) values adjusted in a smoothing operation for instance. An invalid vertex can represent a hole in the structure and can be recovered if the values of its neighbours are known.

3.1 Hole Filling

Reconstructed 3D models normally have a number of holes in the structure. These are caused by noisy reflection of the illuminated surface and by camera and projector occlusion (Robinson *et al*., 2004; Brink, 2008). There are several techniques that can be used to fill in gaps in the mesh such as the ones discussed in Wang and Oliveira

(2003; 2007) and Tekumala and Cohen (2004). In this paper we consider three methods namely bilinear interpolation, Laplace, and polynomial interpolation.

Even though the mesh is defined in 3D, bilinear interpolation (as the name suggests) is performed first in the *x*-direction across stripes and then in the *y*-direction along the stripes. Linear interpolation in the *x*-direction is performed first by simply detecting the number of missing stripes and marking the required indices as valid. The actual (*x,y,z*) values for each vertex $V_{x,y,z}$ are estimated through a simple average of its neighbours' values.

Laplace hole filling in 3D is achieved through iteratively solving Laplace's equation over the surface. An initial value is provided for the missing values and the surface converges to a limit as the number of iterations approaches infinity. We set the number of iterations to 25 to allow real time performance. The way it works can be seen as finding the average of nearby vertices and filling the value of the current vertex (which is the centroid of the area) with the average. Because this is done iteratively over the same area the surface keeps changing until the number of iterations reaches the set value.

Polynomial interpolation works in similar way as bilinear. Instead of filling the missing value with the average of previous and next (say, in the *x*-direction), it uses a 4th order polynomial approximation. In this way, the surface topology is taken into account such that for a missing point on the tip of the nose for example, it would follow the surface of the nose and would fill the missing value with a higher *z*-value than a simple average would do. Thus, in theory it would work better than bilinear and Laplace.

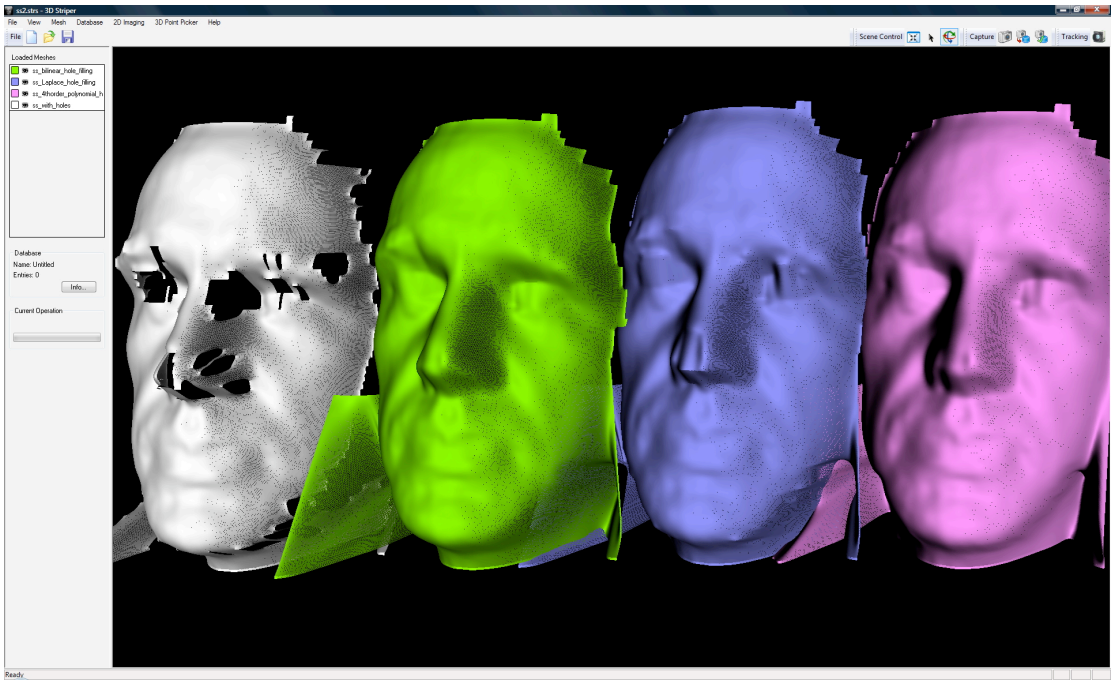

Figure 2: Comparative analysis of hole filling techniques. The white model shows the holes; green: bilinear interpolation; blue: Laplace; pink: polynomial interpolation.

Figure 2 depicts the results of the three schemes discussed. The models have been smoothed using a moving average (see next section) for better visualization and quality assessment. The model (white) was chosen because it has a critical hole located at the tip of the nose, which is notoriously difficult to fill in properly. It is clear that Laplace (blue model) does not work well for the face model as shown on the tip of the nose and the region around the eyes. Bilinear (green) and polynomial (pink) work well on the tip of the nose and it seems that bilinear works slightly better on the region around the eyes.

3.2 Smoothing

Two smoothing techniques are compared namely moving average and Gaussian smoothing. Moving average is performed by cycling first across the stripes *x*-direction) and then along the stripes (*y*-direction). The average of every three points is taken and the middle point is replaced by the average. In this way the boundary vertices remain anchored. Our acquisition method has different resolutions in *x*- and *y*-directions (Robinson *et al*., 2004). Normally, we can resolve one vertex per pixel in the *y*-direction and typically around one vertex per 8 pixels in the *x*-direction. Thus, taking this into account, we performed 8 times more smoothing in the *y*-direction than in the *x*-direction and this has worked well in the face models.

Gaussian smoothing is iteratively estimated by considering 8 neighbours for each vertex. A convolution mask of size 3x3 is applied and the centre vertex is perturbed depending on the values of its neighbours. The difference between each neighbour and the centre vertex is recorded and averaged as ΔV. A multiplier factor is provided (λ) which determines how much of this average should be used to correct the value of the centre vertex; i.e., it defines the momentum of the error correction. We use λ=0.9 and the number of iterations was set to 35. In each iteration cycle *i,* the centre vertex V is corrected by $V_i = V_{i-1} + \lambda \Delta V$.

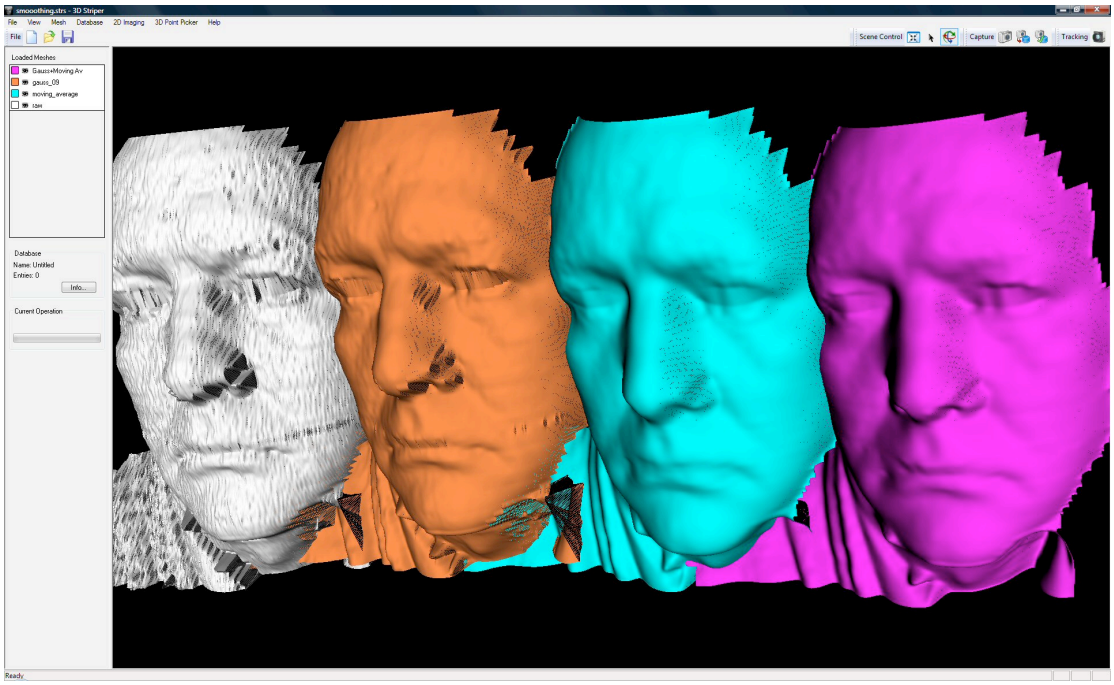

Figure 3: Comparative analysis of smoothing algorithms. The white model shows the original mesh; orange: Gaussian smoothing with λ=0.9 and number of iterations 35; cyan: moving average; magenta: Gaussian smoothing followed by moving average

Figure 3 depicts the results of the smoothing algorithms. It is clear that Gaussian smoothing (orange model) has considerable limitations regarding the perceived quality of the mesh. The moving average algorithm (cyan) seems to work considerably better. By running a Gaussian followed by a moving average seems to slightly improve the model (magenta) especially around the lip area, which becomes more pronounced.

3.2 Noise Removal

The difficulty with noisy data is that for any model it is not straightforward to establish what is structure and what is noise in the data. We observe that the regions around the eyes are normally affected by noise due to the presence of, for instance, heavy make-up delineating the eyes, the presence of eyelashes, and the differential reflectivity of the eyeball. Since we know the position of the eyes in 2D (from 2D face and eye detection) and since a correspondence exists between pixels in the 2D image and the 3D model, the position of each eye in 3D is readily determined.

The question then is how we should go about fixing the eye region. We have tried a number of different schemes and the ones that work best are highlighted here. Since the eyeball is roughly spherical, a natural solution would be to replace the vertices in the eye by a spherical surface centred somewhere behind the face model. By experimentation, we chose the centre of the sphere at a position 40mm behind the face model in the same Z-axis as the centre of each eye. The centres of the spheres for the two eyes are straightforward to determine as the face model is normalized with the origin at the tip of the nose, with the Y-axis parallel to the line connecting the centre of the two eyes. An elliptical mask is marked centred on each eye, and all vertices within the elliptical surface have their values replaced by their spherical counterparts.

A second solution, which is conceptually simpler, is to punch an elliptical hole centred at each eye and then fill in the holes with any of the algorithms discussed earlier. The size of the elliptical mask is dependent on the distance between the eyes to cater for different model sizes. To punch the hole, all that is needed is to mark the vertices in the elliptical surface as invalid, so while they have associated values these would not be considered in any subsequent calculation including for display purposes. When the holes in the eyes are filled, the new values are noted and the points are marked back to valid.

Figure 4 depicts the outputs of the methods. The white model is the original mesh with noisy eyes. The red model shows the effects of the spherical surface replacement; although it works well, the model looks artificial. The yellow model shows the eye holes punched with an elliptical mask and the green model shows eye's hole filling with a bilinear interpolation. Of the models shown, it is clear that the green model seems better than both red and white. While it can be argued (for the models shown in Figure 4) whether or not the green model is a real improvement on

the white, we observed that such improvement can become very accentuated in models with higher levels of noise in the eye region.

Figure 4: Comparative analysis of noise removal around the eyes. The white model shows the original mesh; red: fitting an elliptical mask around the eyes and filling with a spherical surface centred at 40mm behind the model; yellow: punching an elliptical hole around the eyes; green: filling the elliptical hole with a bilinear interpolation.

4. Pose Normalization and Recognition

The mesh is pose-normalized using an iterative method based on the knowledge of three points on the mesh: the location of the eyes and the tip of the nose. The positions of the eyes in 3D are immediately detected from their 2D counterparts from automatic face and eye tracking. The tip of the nose is iteratively detected on the mesh surface from the 3D location of the eyes (that is, the initial point for the tip of the nose is the highest point on the mesh below the eye positions).

The origin is then translated to the initial tip of the nose. Figure 5 shows the 3 axes where green is the z-axis, yellow is the y-axis, and red is the x-axis. First, the mesh is rotated around its z-axis (green) such that the x-axis is aligned with the y-coordinates of the two eyes (that is, the two eyes must have the same y-values). A second rotation is performed around the y-axis (yellow) such that the x-axis is aligned with the z-coordinates of the two eyes (the two eyes must have the same z-values). Finally, the last degree of freedom is removed by rotating the mesh around its x-axis (red) such that the y-axis is at a constant angle value in relation to the z-value of the two eyes (45 degrees in this case). This process is repeated a number of times (maximum 3) or until the changes in rotation are below a set threshold.

Once the model is pose-normalized, that is, rotated and translated to a standard position then a number of planes are cut through the face model in pre-determined ways and the intersection of these planes with the mesh mark the feature points of

interest. We have chosen to extract 43 feature points located on the more rigid region of the face as shown in Figure 5. Then the normalized distances between such points are estimated and represented by a vector of distances. These vectors of distances uniquely characterize a face model in the database and form the basis for identification and verification.

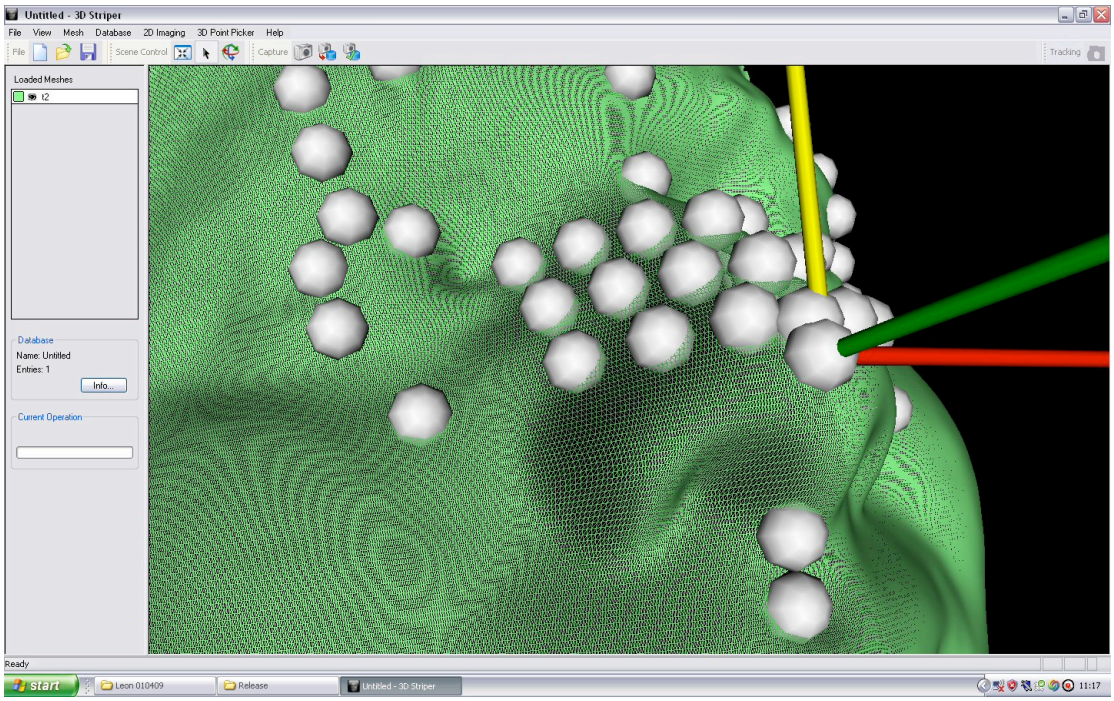

Figure 5: Pose normalization and 3D feature extraction.

The recognition method is based on eigenvector decomposition. All vectors of distances are placed in the database in matrix format where each row represents one measurement and each column represents one face. First the scatter matrix of the database is calculated followed by the estimation of the co-variance matrix, which measures the tendency of two vectors (two faces) to vary together in the database. The eigenvectors are calculated and sorted by eigenvalues. Only the first 15 eigenvectors are selected corresponding to the 15 largest eigenvalues. A linear combination of these eigenvectors is used and kept in a *Matrix of Weights*, which forms the basis for recognition.

Once the matrix of weights is obtained, enrolled and to be verified face models are subject to this matrix transformation. Identification is a one-to-many search and is achieved by measuring the shortest multi-dimensional distance from a subject's feature vector to every other vector in the database. The shortest distance points to the identity of that person in the database. In verification scenarios a one-to-one search is performed in a process that relies on a threshold for maximum distance being specified and this is set through a user-interface. In Figure 1 above, the thumbnails represent the current image and the three closest matches in the database with a green banner displaying PASSED:0 and THRESHOLD: < 500. The parameter PASSED represents the match with a value of zero being a perfect match. If the parameter were greater than the set THRESHOLD, then a red banner would display FAILED with the corresponding measure of distance.

We tested the method with 65 subjects in controlled acquisition where four models were generated for each person: two models with high density mesh and two with low density, where a high density mesh has exactly the double the number of vertices as compared to its low density counterpart. We enrolled one set (65 high + 65 low) and used the remaining for testing purposes. It is important to stress here that the four models were generated from two distinct 2D images, where the models from one image were used to enroll in the database and the models from the other image were used for testing purposes. An interesting observation was made that all high density models had their closest match to another high density model while the low density model had closest match to both. We had 100% accuracy for the high density models while 6/65 failures for low density models (91% accuracy). The accuracy obviously reflects the particular set of measurements taken and the relative influence of a subset on overall performance. Despite lower accuracy rate, the low density models cannot be discarded at this stage as it suggests that it can improve in the long run for large databases.

5. Discussion and Conclusion

We have presented an overview of a process model for real-time 3D face recognition. It was not the purpose of the paper to comment in details on all stages of the process rather, to update on recent developments concerning real-time tracking and detection in 2D and some of the required 3D post processing operations. In particular, we focus on hole filling, smoothing, and noise removal.

Concerning hole filling, the Laplace method can be ruled out given its poor ability to correctly fill holes in the smooth surface of a face model. We pointed out that bilinear and polynomial interpolation seems to be equivalent in terms of quality of the output model. However, the bilinear interpolation has fewer operations to complete than polynomial interpolation and thus, it is more appropriate for real-time operation and it is to be preferred.

Concerning mesh smoothing, Gaussian smoothing on its own is substantially poorer than a simple moving average algorithm. However, moving average tends to smooth out some important features in the vertical direction; an example is shown where the lips got over-smoothed and lost some of the features. We also tried a combination of Gaussian and moving average and this yielded the best results. For applications in 3D face recognition we still believe that moving average is preferable given that it is slightly faster than Gaussian as it performs fewer operations, and some of the lost features have been observed not to impair recognition. The combination of both remains an option as this can still provide real-time performance. We have briefly presented other necessary operations such as pose normalization, feature extraction and recognition and presented results on classification. We have tested high and low density meshes and the first results indicate that high-density meshes, despite their computational and storage costs are preferred.

We have tested the above algorithms in a real-time scenario using a Sony Vaio VGN AR61ZU with Intel Duo Core, 2.4GHz, 4GB memory. All pre-processing operations in 2D (face and eye tracking, image filtering, stripe indexing) and 3D post-processing (generation of point cloud and triangulation, noise removal, hole filling, smoothing,

subdivision, normalization, feature extraction and recognition) are performed in 1s 200ms from the point where the tracking locks on the two eyes. This has been tested using the bilinear hole-filling, moving average smoothing and elliptical mask eye fixing with bilinear interpolation.

We stress that all the above operations are included in the overall 1s 200ms per model, which is a remarkable performance. It is also important to stress that in order to achieve this level of performance, 3D models are never displayed, only 2D face and eye detection video stream are displayed. Normally we use VTK to handle the display of 3D models and this process consumes over 2 seconds per model and, in a continuous cycle it can take up to 5 seconds per model.

In conclusion, we have demonstrated real time performance of a 3D face recognition system and fulfilled our claims in Section 1for a system that can be used in a realistic scenario. The developed system has been implemented using a small database (130 entries). Future research will address bettering the quality of the models, the porting to a self contained unit that is able to operate in the near-infrared spectrum, and testing of the method in very large databases (thousands of subjects).

References

Adolf, F. (2003). How-to build a cascade of boosted classifiers based on Haar-like features. http://lab.cntl.kyutech.ac.jp/ kobalab/nishida/opencv/OpenCV ObjectDetection HowTo.pdf.

Bowyer, K.W., K. Chang, and P. Flynn (2004). A Survey Of Approaches To Three-Dimensional Face Recognition, Int Conf on Pattern Recognition (ICPR), 358–361.

Bradski, G.R and V. Pisarevsky (2000). Intelapos' Computer Vision Library: applications in calibration, stereo segmentation, tracking, gesture, face and object recognition. Computer Vision and Pattern Recognition. Proceedings. IEEE Conference on Volume 2, 796 – 797.

Brink,W. (2008). 3D Face Scanning and Alignment for Biometric Systems, PhD Thesis, Sheffield Hallam University.

Brink, W., A. Robinson, M. Rodrigues (2008). Indexing Uncoded Stripe Patterns in Structured Light Systems by Maximum Spanning Trees, British Machine Vision Conference BMVC 2008, Leeds, UK, 1–4 Sep 2008.

Cook, J., C. McCool, V. Chandran, and S. Sridharan (2006). Combined 2D/3D Face Recognition Using Log-Gabor Templates, Advanced Video and Signal Based Surveillance, IEEE Conference on, pp. 83, 2006 IEEE Intl Conf on Advanced Video and Signal Based Surveillance (AVSS'06).

FRGC, (2005). The Face Recognition Grand Challenge, http://www.frvt.org/FRGC/

GMPR, 2009. Geometric Modelling and Pattern Recognition Video Demos at http:// www.shu.ac.uk/ research/meri/gmpr/videos.html

Gordon, G. (1992). Face recognition based on depth and curvature features. Computer Vision and Pattern Recognition (CVPR), 108–110.

Medioni, G. and R.Waupotitsch (2003). Face recognition and modeling in 3D. IEEE International Workshop on Analysis and Modeling of Faces and Gestures (AMFG 2003), 232–233.

Hesher, C., A. Srivastava, and G. Erlebacher (2003). A novel technique for face recognition using range images. 7th Int Symposium on Signal Processing and Its Applications).

Rodrigues, M.A. A. Robinson,W. Brink (2008). Fast 3D Reconstruction and Recognition, in New Aspects of Signal Processing, Computational Geometry and Artificial Vision, 8th WSEAS ISCGAV, Rhodes, 2008, p15–21.

Rodrigues, M.A., A. Robinson, W. Brink (2007). 'Issues in Fast 3D Reconstruction from Video Sequences, Lecture Notes in Signal Science, Internet and Education, Proceedings of 7th WSEAS International Conference on MULTIMEDIA, INTERNET and VIDEO TECHNOLOGIES (MIV '07), Beijing, China, September 15-17, 2007, pp 213–218.

Rodrigues, M.A., A. Robinson, L. Alboul, W. Brink (2006). 3D Modelling and Recognition, WSEAS Transactions on Information Science and Applications, Issue 11, Vol 3, 2006, pp 2118–2122.

Robinson, A., L. Alboul, and M. Rodrigues (2004). Methods for indexing stripes in uncoded structured light scanning systems. Journal of WSCG, 12(3) 371–378, February 2004.

Robinson, A., M.A. Rodrigues, L. Alboul (2005). Producing Animations from 3D Face Scans,Game- On 2005, 6th Annual European GAME-ON Conference, De Montfort University, Leicester, UK, Nov 23–25, 2005.

Seo, N. (2009). Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features), http://note.sonots.com/ SciSoftware/haartraining.html

Tekumalla, L.S., and E. Cohen (2004). A hole filling algorithm for triangular meshes. tech. rep. University of Utah, December 2004.

Wang, J. and M. M. Oliveira (2003). A hole filling strategy for reconstruction of smooth surfaces in range images. XVI Brazilian Symposium on Computer Graphics and Image Processing, pages 11–18, October 2003.

Wang, J. and M. M. Oliveira (2007). Filling holes on locally smooth surfaces reconstructed from point clouds. Image and Vision Computing, 25(1):103–113, January 2007