Face Replacement Demo using the Kinect Depth Sensor

Jared Sanson
Department of Electrical and Computer Engineering
University of Canterbury
Christchurch, New Zealand
jared@jared.geek.nz

Dr. Richard Green
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
richard.green@canterbury.ac.nz

***Abstract*— This paper proposes a face-swap method, wherein the use of a depth sensor and improved algorithms are used to improve the quality and realism of a face swap process. By tracking head pose and facial features in 3D using a Kinect depth camera, an accurate model of the face can be constructed and used to deform a texture which is then drawn on top of a 2D video stream. The use of random regression forests for head pose estimation is explored, along with a novel method for blending textures based on luminance. The proposed system shows good results through a wider range of head rotations than previous methods, and is able to run at an average of 25fps.**

Keywords — depth sensor; face tracking; pose estimation; kinect; candide; texture warping; face replace

I. INTRODUCTION

The goal of the proposed method is to improve upon a face-swap method in prior research [2-7] by investigating the use of a structured light depth camera, and the use of algorithms more suited to this type of data. A novel method for blending textures based on luminance is also explored.

The original method would swap people's faces as they walked past a display, and demonstrated many common algorithms for face detection and recognition. It assumed a simple flat-replacement model with no rotation of the face texture so it would only work well if the user was facing directly towards the camera. This can be improved upon by tracking the user's face in 3D and utilizing a model of the face to project the face overlay texture, which will warp the texture to match the actual face.

Much work has already been done on the face-swap system, mainly looking at improving performance and more advanced blending techniques [2, 6, 7]. Instead of improving on these methods, this paper proposes an entirely different approach using algorithms more suited for depth camera imagery.

II. BACKGROUND

A. The Face-Swap System

The original face-swap system requires various algorithms; the core consists of a face-detection algorithm such as the Haar-Cascade, which is used to locate the face in the image and to know where to overlay the swapped face. Object tracking algorithms such as Camshift or Optical Flow help improve performance by removing the need to re-detect the face in every time. It can also help smooth out the movement between frames, as face detection algorithms tend to be imprecise in locating the exact location of the face.

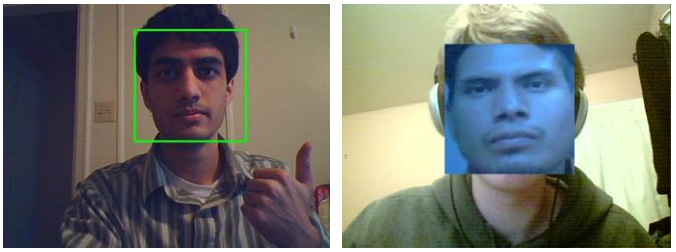


Figure 1. Results from two previous face-swap papers [4, 5]

After the face is localised, the face texture can be extracted from the video and stored in a database, and a new face can be overlaid on top of the video feed in real-time. This is done by simply resizing the texture to match the bounding box of the detected face, although this does not work well if the face rotates since the bounding box does not rotate with it.

To prevent swapping a user's face with their own a face recognition algorithm such as Eigenfaces or Fisherfaces must be used, which compares the detected face against a database of faces to ensure the two are different.

B. Previous Work

The first student paper performed on this subject [2] implemented the Haar-Cascade algorithm for face recognition, but only used a simple time-based averaging algorithm for tracking, and had no face recognition at all. The averaging did provide some resilience to dropped frames but because it had to detect the face for each new frame it had reduced performance. It did however attempt to adjust the face overlay colour by

analysing a histogram of skin coloured pixels, which could potentially be adapted for the method proposed in this paper.

This was improved upon by subsequent papers [3, 4] which added face recognition using the Eigenfaces algorithm, and object tracking using Camshift. These modifications significantly improved performance and reliability, although the initial face detection stage still took a large amount of time to process. The Camshift algorithm additionally allowed the face texture to be rotated in the plane of the screen.

One paper looked at replacing the face recognition algorithm with Local Binary Patterns (LBP), which improved the running performance, but offered no improvements for the initial face detection stage [7].

Another paper looked at improving the performance of the initial face detection by implementing the Viola-Jones face detector algorithm using a highly parallel GPGPU, which showed an improvement of up to 300% [6]. It also investigated the use of Fisherfaces for face recognition, which has the advantage of being invariant to changing lighting conditions. It improved the reliability of tracking and produced less false positives than previous methods, and additionally attempted to improve the face overlay blending by utilizing a radial blur filter to smooth the edges of the face texture. The radial filter produced very good results, but only in the case where the lighting conditions and head rotation were similar.

Figure 2. Face replace using a radial blur filter [6]

C. Depth Sensors

There are various methods for acquiring 3D depth data which should all perform well for the methods outlined in this paper. Structured-light is one such technology that has become rather popular, due to the emergence of cheap depth cameras such as the Microsoft Kinect or the PrimeSense camera. These cameras do not perform as well as more sophisticated cameras and fail in bright sunlight, but their low cost provides a huge advantage for researchers.

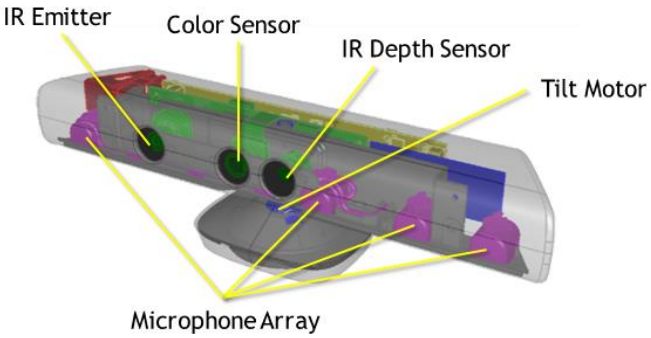


Figure 3. The Kinect depth camera [1]

Adapting the existing face-swap framework to utilize a structured-light depth sensor allows the use of much more advanced algorithms specifically designed for RGB+depth data. Some of the commercial offerings that implement such algorithms include Microsoft's Kinect Developer SDK [1] which is designed for use with their Kinect depth sensor, and Visage Technologies' face tracking library [8] which operates on regular 2D imagery. Unfortunately the algorithms used are proprietary and details about their implementation are not available, so this paper will outline published algorithms that achieve a similar result.

D. Head Pose Estimation

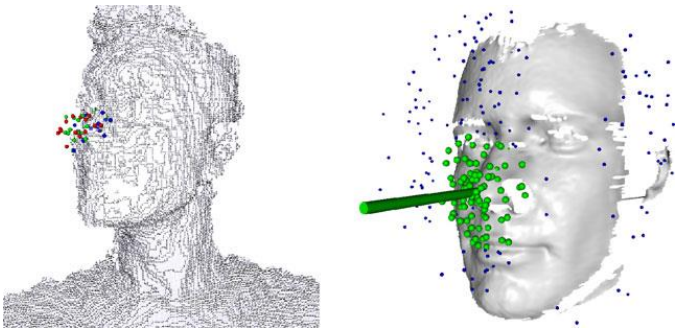


Figure 4. Head Pose Estimation [9]

There are a few various algorithms that can be used to determine the orientation of the head in 3D space, which generally use machine learning techniques [9, 10] such as use of neural networks and random regression forests, in order to determine a rough orientation estimate. While the Kinect face tracking library does not outline the specific algorithm it uses, it is quite possible it uses neural networks or regression forests as Microsoft have published a paper which uses these methods for detecting a skeleton model [11].

Another paper proposes a more analytical method which involves solving a large matrix, and has the advantage that it does not require prior training [12]. This method was able to run in real-time, with the head between 0.6 to 6 meters from the camera.

E. Candide Face Model

The Candide model is widely used for various research, as it provides a simple mesh with standardised features and defines a robust way to deform the mesh based on supplied parameters. The latest version is Candide-3, and implements 113 vertices, 65 animation units representing various facial features, and 14 shape units [13]. It also contains a subset of the MPEG-4 FBA specification [14], which is intended to provide a way to animate various facial expressions.

The Shape Units allow the Candide-3 mesh to adapt to any user's face, such as by changing the distance between the eyes, or the width and height of the mesh.

F. Face Expression Tracking

In order to control the Animation and Shape Units in the Candide-3 model, the facial features must be extracted from the given video frame. This is usually done in 2D on the colour image stream [8] by locating important facial fiducial points, however some algorithms have been devised which operate on depth imagery instead [15].

There are many methods that can be used for locating these facial feature points, such as the Haar classifier as used in Tresadern, et al. [16], or through stochastic methods such as those used in Dornaika and Davoine [17].

Figure 5. Tracking and recognition of facial features[17]

The Kinect library also provides a robust facial feature tracking algorithm designed specifically for the Candide model, however the exact methods used are not published. It only provides 6 Animation Units and 11 Shape Units, which is not enough to control all of the facial animation parameters, but it is enough to show basic deformation of the mouth, chin, and eyebrows.

G. Limitations

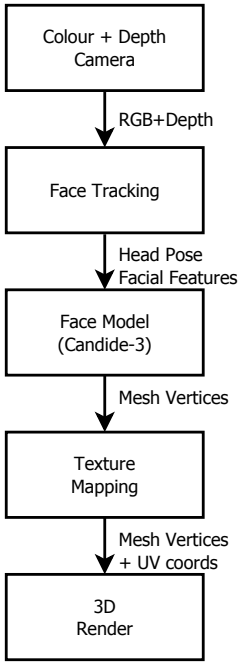
There are a few limitations that apply to the current work done on the 2D face-swap system, which this paper hopes to solve. Currently these methods assume the face is a flat rectangle, with the bounds detected through the use of a face detection algorithm such as the Haar Cascade classifier or the Viola Jones classifier. These algorithms do not provide any information about the shape of the face being detected, nor the orientation and pose of the face in 3D space.

Because of this assumption, the texture cannot be realistically blended in all orientations of the face, especially for extreme rotations which introduce occlusion and significant texture warping. This cannot be fixed by improved blending techniques such as the radial blur filter [6], because it assumes the user is looking directly towards the camera with no rotation in 3D space.

III. Solution

A. Overview

The methods outlined above are combined in order to build a new face-swapping demonstration system, using a structured light depth camera instead of a regular colour camera. The proposed method is restricted to using a pre-defined face overlay texture and face model.



The proposed method consists of a head tracking and pose determination algorithm, but does not implement face recognition like the original method as the face overlay texture is pre-defined. After determining the head pose and location, facial features are extracted which are later used to deform the face model.

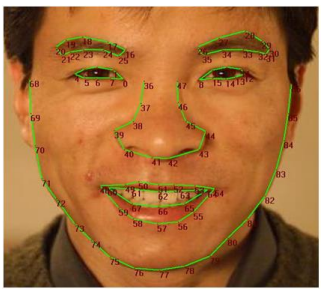
The Candide-3 model is used as the face model, which can be deformed by head pose, animation units detected from the facial features, and shape units which represent the shape of the detected face. These deformations allow a single mesh to take the shape of any detected face.

A face texture can then be mapped to the model so that it deforms to match the tracked user, and an advanced luminance-blend algorithm used to blend the texture onto the underlying video frame.

Finally, the texture mapped face mesh is passed onto a 3D rendering engine which can project the face model with the assigned texture onto the camera image plane, thus overlaying the face onto a tracked user.

B. Face Tracking

The Kinect Developer SDK is used for detecting and tracking faces within video obtained from a depth sensor. It operates on both RGB and depth imagery to obtain the head pose, head location in 3D space, 2D facial feature points (shown in the figure to the right), and Candide-compatible animation and shape units. The rotation and location alone are enough to implement the face swap method, although implementing shape units will allow



[1] Figure 6. 2D facial features returned by the Kinect library [1]

the face model to match different head shapes, and animation units will allow the face model to match the user's facial expressions.

The initial search for a face is a lengthy operation, so the Kinect library provides a separate tracking algorithm which performs much less processing once a face is found in the image. This allows for much higher frame rates during active use, in much the same way the original face-swap methods used Camshift to prevent having to detect a face every frame.

Generally it takes a certain amount of time before shape units are available to the application, presumably because the library needs to learn the shape of the tracked face. Once these values are available, the face model can be updated to match the user's personal head shape and size.

The biggest limitation of the Kinect Face Tracking Library is that it only reports 6 animation units, which only provides enough information to animate the lips, eyebrows, and lower jaw. The MPEG-4 FBA specification [14] defines 66 Facial Animation Parameters (FAPs) in total, so other libraries and algorithms may be able to detect a much larger set.

Another problem with the Face Tracking Library is latency and jitter in the returned data, which causes some alignment issues of the face overlay when the user is moving. The jitter can be reduced through the use of a temporal filtering function such as a low pass filter or a kalman filter, and the latency could be improved by improving the underlying library or augmenting it with other computer vision techniques such as optical flow or the Camshift algorithm.

C. Face Model

The face model is implemented using the WFM file format, which provides a way to define vertices, faces, animation units, shape units, texture co-ordinates, and a global transformation. However for simplicity, the Candide-3 model is used as it is a well-established model used in many research papers [13], although the texture co-ordinates must be defined manually. The Candide-3 model defines 65 animation units and 14 shape units, but not all of these are actually used or required.

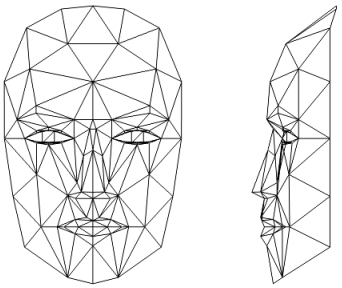


Figure 7. The Candide-3 model

AU/SU deformation of the mesh is optional, since just the head rotation and translation can be enough to transform the face model to align properly with the user's face. However the use of SUs to deform the mesh could potentially allow the system to work over a much larger variety of faces, and the use of AUs can modify the texture to make it appear to match the user's facial expressions.

D. Texture Mapping

For best results the desired face texture can be manually mapped to the Candide-3 mesh, so that the various facial features are properly aligned to the texture. Alternatively, the texture can be automatically extracted from the current video frame, using the current face mesh overlay to map the texture to the Candide-3 vertices. This technique may fail if the head is rotated, due to obscured regions in the face, so a manual texture mapping is used for this paper.

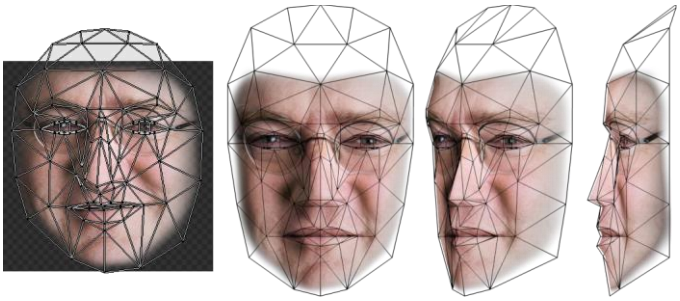


Figure 8. Manual Candide-3 Texture Mapping

The figure above shows the process of manually mapping UV texture co-ordinates of the Candide-3 model to a 2D face texture image. The left view shows the 2D texture co-ordinates with respect to the texture, while the 3 views on the right show the texture applied to the 3D Candide-3 mesh.

E. Blending

In order to provide a smooth blend of the face texture onto the video stream, the texture is manually blurred at the edges and stored as an RGB+Alpha image. This improves the blending around the edges of the face overlay, by allowing the face overlay texture to smoothly blend into the underlying video frame.

However if lighting conditions differ from the original overlay image, the overlay will not match the video and it will not blend smoothly. To help improve blending under differing lighting conditions, we can employ a luminance blend followed by automatic brightness correction, as shown in the following figure:

Figure 9. Left – naïve alpha blend, middle – luminance blend, right – luminance blend with automatic levels correction.

The luminance blend combines the luminance (brightness) of the face texture with the chrominance (colour) of the underlying video frame. This works because most of the perceptible detail in an image lies in the luminance channel [18], so the result is that the face takes the colour of the underlying video while retaining the detail. This is especially useful for cameras with poor white balance correction, such as in the figure above which makes the face appear to have a purple tint.

Luminance blending is performed in a luma-chroma colour space using an encoding such as Y'CbCr which splits the luminance and chrominance into perceptually meaningful information. For this application the conversion between RGB and Y'CbCr as defined by the JPEG standard is used[19]:

$$\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix}$$

The exact equation used is not important, as long as it converts between RGB and luma-chroma colour space. The following figure illustrates the effect of converting between RGB and Y'CbCr encodings:

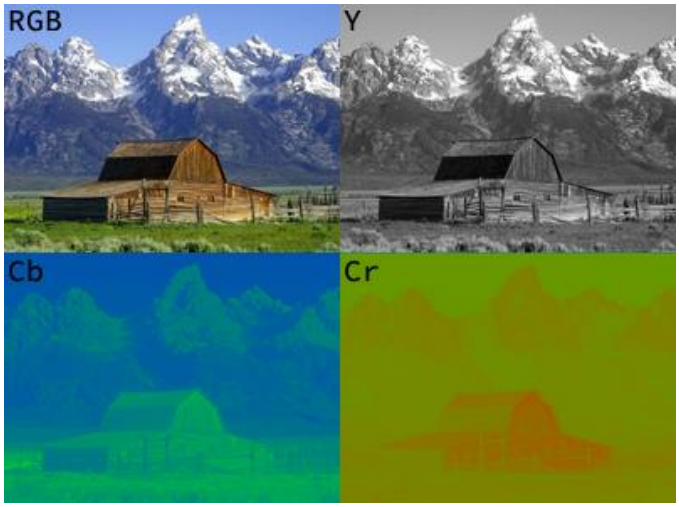


Figure 10. Y'CbCr components[1]

Generally it is not desirable to use a colour space such as HSV, because the value component does not correspond to human perception of luminosity (in HSV colour space all fully saturated colours are defined to be the same brightness, when in reality green is brighter than blue). It is also not trivial to convert between RGB and HSV colour spaces, and involves a lot of conditional calculations.

Luminance blending alone is not sufficient, as differences in brightness can show up if the video is substantially brighter or darker than the face overlay. To compensate for this, the underlying face in the video frame is analysed to compute the minimum and maximum brightness, and the face overlay texture compensated to match accordingly. This helps keep the blend looking consistent in a wide range of lighting conditions.

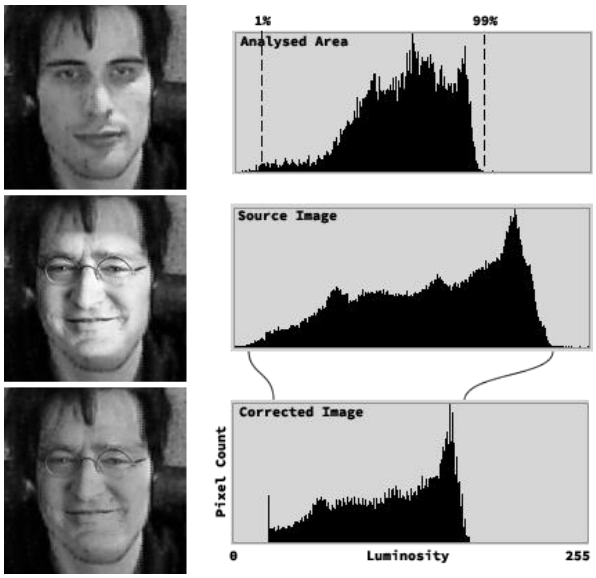


Figure 11. Brightness correction using luminance histograms

This is done by analysing the luminance histogram of the underlying face region in the video, which counts how many times a pixel value occurs in the image. To find the minimum and maximum brightness, the pixel counts are summed to find the total count, and then accumulated to find the 1% and 99% points in the histogram. This helps provide some robustness against noise, in case there are a few counts of completely black or white pixels. To simplify calculations, this assumes the source image is normalised.

The only limitation remaining is that these methods cannot compensate for differing shadows, for example if the light source is located on the opposite side of the face overlay image, or if a shadow falls onto the face. This could perhaps be compensated for by blurring the underlying face image, and blending the luminance channels of both the face and the video to attempt to add shadows to the face overlay texture, although this was not explored in this paper.

For efficiency and flexibility, this process could be performed using GLSL shaders, which run directly on the GPU and are well suited for this type of processing. The use of Y'CbCr colour space serves well for this kind of work, as the transformation between RGB and Y'CbCr can be done quickly and efficiently with a single matrix multiply, and the brightness correction values can be passed in as parameters.

F. Rendering

In order to overlay the 3D face model on top of the 2D video feed, the 3D model must be projected onto the plane of the camera using a perspective transform. This can be modelled with a simple pin-hole camera model, which defines the camera's focal plane as a function of its sensor size and focal length. This model is shown below:

[1] http://commons.wikimedia.org/wiki/File:Barn-yuv.png

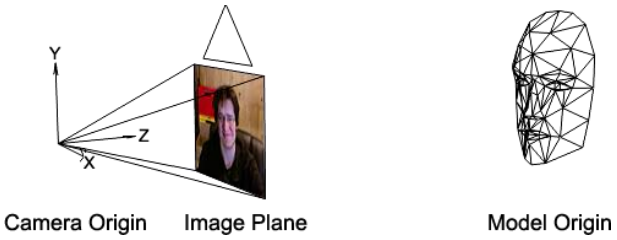


Figure 12. Pinhole camera model

The deformed 3D face model's origin is defined to be at the centre of the head, so it must be rotated and then translated in order to put it in world space. After this a perspective projection matrix can be applied to project the 3D model onto the image plane.

Rotation of the face model returned by the Kinect Face Tracking Library is defined by a set of three intrinsic euler rotations (α, β, γ):

$$M_R = Z(\gamma)Y(\beta)X(\alpha) =$$

$$\begin{bmatrix} cos\gamma & sin\gamma & 0 & 0 \\ -sin\gamma & cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & -sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ sin\beta & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & sin\alpha & 0 \\ 0 & -sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And translation is defined as meters from the camera's origin:

$$M_T = \begin{bmatrix} 1 & 0 & 0 & T_X \\ 0 & 1 & 0 & T_Y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The Kinect camera has a small offset between the camera origin and the world origin, however for this application it can be ignored it as it does not introduce much error. Thus, eye-space and world-space are identical and no transformation between the two is required.

To construct the perspective projection matrix, the intrinsic camera parameters must be known. For the Kinect camera, the focal length is approximately 87.64mm at a resolution of 640x480. Using these values we can construct the perspective projection matrix[20]:

$$M_P = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_f + z_n}{z_n - z_f} & \frac{2z_f z_n}{z_n - z_f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where f is the focal length, $aspect = \frac{width}{height}$, z_n is the near clipping plane, and z_f is the far clipping plane. The near and far clipping planes are used for depth testing within the OpenGL pipeline[20], and for best results should be set to the closest and furthest distances the Kinect can measure.

Finally to render the 3D model onto the image plane, the required transformations are applied to every vertex of the face model in order to convert them to Normalized Device Space (NDC):

$$\begin{bmatrix} X_{clip} \\ Y_{clip} \\ Z_{clip} \\ W_{clip} \end{bmatrix} = M_P \times M_T \times M_R \times \begin{bmatrix} X_{model} \\ Y_{model} \\ Z_{model} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{NDC} \\ Y_{NDC} \\ Z_{NDC} \end{bmatrix} = \begin{bmatrix} X_{clip}/W_{clip} \\ Y_{clip}/W_{clip} \\ Z_{clip}/W_{clip} \end{bmatrix}$$

The process of mapping world co-ordinates to clipping co-ordinates is shown below:

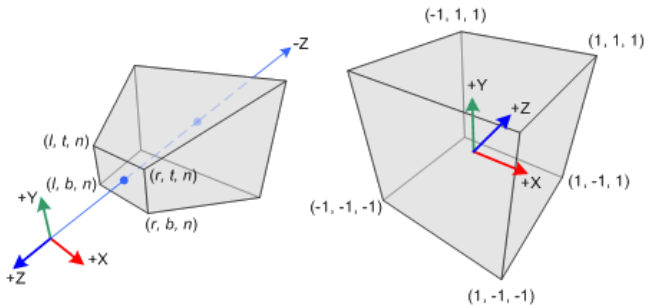


Figure 13. OpenGL Perspective Projection [20]

IV. RESULTS

Figure 14. Final result showing luminance blend, automatic level adjustment, and texture warping

Figure 15. Result showing inaccuarcy in texture to mesh mapping - the nose has deformed incorrectly

Figure 16. Result showing the effects of texture warping and 3D projection

A. *Hardware & Software*

The solution was implemented in VC++ using the Kinect Face Tracking Library [1], with OpenGL and GLSL shaders for rendering, and OpenNI for the depth sensor interface. OpenNI allows the demo software to be used with any compatible depth camera, although the Kinect Face Tracking Library may restrict use to Kinect cameras only.

A Kinect depth sensor was used as the input device and the demonstration program was run on an Intel i5 3.3GHz CPU with an NVIDIA GPU for OpenGL rendering. The demonstration program was compiled with full optimisations and no debug information, to maximise performance. Performance is significantly reduced if it is compiled in debug mode.

B. *Quantified Results*

The demonstration program ran at between 15-30FPS while detecting a face, which is close to the 30FPS update rate of the Kinect camera, and similar to previous research[7] [6].

Limits:

Minimum Detection Distance	550mm
X Rotation (Pitch)	-40° to 40°
Y Rotation (Yaw)	-40° to 40°
Z Rotation (Roll)	-80° to 80°

C. *Limitations*

The implemented solution is limited in the range of face poses it can track, as it is only trained to detect faces that are looking towards the camera. It tracks best when the head pitch is less than 20°, roll is less than 90°, and yaw is less than 45°. This is a limitation of the proprietary library used, but could potentially be improved by training a custom implementation on a larger set of head poses. It also exhibits poor tracking when the head rotates close to these limits.

Heads must also not be too far or too close, or they cannot be tracked. This is partially due to the reduced resolution of the depth imagery at close and far ranges, but is actually more of a limitation of the training set.

The tracked head pose also shows some jitter and latency, because no filtering or interpolation is performed on the data. Tracking latency is particularly noticeable when the head moves across the frame, and results in misalignment between the face overlay and the tracked user.

Another limitation of this proposed method is the absence of any face recognition or face texture extraction algorithms, such as what was used in previous work. As such it must rely on manually provided face textures to overlay onto the user's face, and the texture must be manually mapped to the Candide-3 face model.

V. Conclusion

The proposed method provides a much more realistic face overlay than what could be done with regular 2D algorithms, while operating with around the same level of performance and reliability. The improved blending technique allows a face overlay texture to work in different lighting conditions, and the face model allows the face texture to be warped to match the tracked user's face.

A. Future Work

There are many improvements that could be made in future research to re-implement the original functionality provided in existing face-swap work, such as automatic face texture extraction and face recognition. The proposed method only supports tracking of one user at a time, but it could be modified to support multiple people, and the functionality for swapping two people's faces could be re-implemented into this new solution.

Currently the proposed method must be set up with custom face textures manually mapped to the Candide-3 model, but this could be made automatic through face feature detection and texture extraction methods.

Another improvement could be to convert the video stream into a 3D point cloud using the Point Cloud Library (PCL), which would allow much more advanced techniques for overlaying face textures and would reduce the disparity between the face overlay and user's face. This would also make it relatively easy to enable foreground objects to occlude the face overlay, which currently is not possible.

Finally, the use of the proprietary Kinect library could be replaced with well understood head pose and facial feature detection algorithms, or even with algorithms suited for 3D point cloud data. There may be large performance and reliability improvements available by doing this.

VI. References

[1] Microsoft Developer Network. (2013). *Kinect Face Tracking SDK*. Available: http://msdn.microsoft.com/en-us/library/jj130970.aspx

[2] Z. Zheng and R. Green, "Face Replace," University of Canterbury, 2009.

[3] M. Elmadani and R. Green, "Face Replace," University of Canterbury, 2011.

[4] H. Jenkins and R. Green, "Improved Method of Face Replacement," University of Canterbury, 2011.

[5] L. Chin and R. Green, "Improved Face Replace Demo," University of Canterbury, 2013.

[6] M. Lancaster and R. Green, "Enhanced real time facial detection and replacement using GPGPU," University of Canterbury, 2013.

[7] M. Smith and R. Green, "Improved Face Replacement Demo," University of Canterbury, 2013.

[8] Visage Technologies. (May 2014). *FaceTrack SDK*. Available: http://www.visagetechnologies.com/products/visagesdk/facetrack/

[9] G. Fanelli, J. Gall, and L. Van Gool, "Real Time Head Pose Estimation with Random Regression Forests," 2011.

[10] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Gool, "Random Forests for Real Time 3D Face Analysis," *International Journal of Computer Vision*, vol. 101, pp. 437-458, 2013.

[11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, *et al.*, "Real-Time Human Pose Recognition in Parts from Single Depth Images," *Microsoft Research Cambridge & Xbox Incubation*, 2011.

[12] F. A. Kondori, S. Yousefi, L. Haibo, S. Sonning, and S. Sonning, "3D head pose estimation using the Kinect," in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, 2011, pp. 1-4.

[13] J. Ahlberg. (2001). *Candide-3 - An Updated Parameterised Face*. Available: http://www.icg.isy.liu.se/candide/

[14] "MPEG-4 Face and Body Animation (MPEG-4 FBA) - An Overview," ed: Visage Technologies AB 2012.

[15] S. Gupta, M. K. Markey, and A. C. Bovik, "Anthropometric 3D Face Recognition," *International Journal of Computer Vision*, vol. 90, pp. 331-349, 2010.

[16] P. Tresadern, M. Ionita, and T. Cootes, "Real-Time Facial Feature Tracking on a Mobile Device," *International Journal of Computer Vision*, vol. 96, pp. 280-289, 2012.

[17] F. Dornaika and F. Davoine, "Simultaneous Facial Action Tracking and Expression Recognition in the Presence of Head Motion," *International Journal of Computer Vision*, vol. 76, pp. 257-281, 2008.

[18] S. Sudhakaran. (2013). *Understanding Luminance and Chrominance*. Available: http://wolfcrow.com/blog/understanding-luminance-and-chrominance/

[19] C.-C. Microsystems. (1992). *JPEG File Interchange Format*. Available: http://www.w3.org/Graphics/JPEG/jfif3.pdf

[20] S. H. Ahn. (2012). *OpenGL Projection Matrix*. Available: http://www.songho.ca/opengl/gl_projectionmatrix.html

Images of Gabe Newell used in this paper are public domain.

Source code available: https://github.com/jorticus/face-replace