Face It: 3D Facial Reconstruction from a Single 2D Image for Games and Simulations

J. Steven Kirtzic, Ovidiu Daescu
Department of Computer Science
University of Texas at Dallas
Richardson, TX USA
jsk061000, daescu@utdallas.edu

Abstract—An ongoing challenge with game and simulation design is immersion. Users want to feel as though the avatars that they see on screen are themselves and that the experiences of their avatars are their own. To that end, much work has been done in the area of 3D facial reconstruction for the purpose of inserting one's own likeness into a game or simulation. However, most existing methods require a minimum of two images and complex computations to produce the 3D head mesh. In this paper we propose a simple method for 3D facial reconstruction that renders the face mesh as a terrain using a single input image and minimal computation. This results in a fast and lightweight facial reconstruction that is highly portable and has a wide variety of applications.

1 Introduction

3D facial reconstruction is not a new problem. Indeed, the problem has been studied for many years, with a wide spectrum of solutions having been developed [1] [2] [3] [4]. The goals with the current research in this area vary according to one's application. In some applications, particularly academic, the goal is to develop a system that creates the highest-quality, most realistic 3D facial reconstruction, regardless of the time and computational power required to do so. However, in many other applications, particularly within industry, time and power are more of a factor, so quality is often sacrificed for the sake of speed and convenience.

The video game and simulation field is one that is of considerable interest today, not just in industry but also in academia. Research in this area is not only fueled by the desire to capitalize on it's popularity and profitability, but also by the desire to expand its potential into areas and applications that were once thought to be unreachable and intractable. This development has led to increasing degrees of immersion into the game or simulation environment, to the point that common consumer game consoles no longer even require the player to use a handheld controller. As we continue this work into deeper levels of immersion the virtual worlds and lives that we create become more and more intertwined with our real world and lives. One aspect that is returned to again and again is the ability of the user to insert their own likeness into the game or simulation, typically with their own visage as that of their avatar in the game. Consequently, there have been several games released for console systems as well as other applications that allow the user to do just that - replace their character's face with a 3D rendering of their own. While the processes used by these games and applications achieve the required result, they are often relatively slow and cumbersome, overly complex, or use unnecessarily large software components which limits their portability. Gamers, as well as consumers in general, have little patience for such slow and complex systems, and tend to prefer products that produce immediate results at a minimal cost. Also, as the proliferation of smart phones continues to grow, as does their computing power, users are increasingly using their phones for a multitude of applications including games and other immersive activities. Thus, we propose a novel system designed to provide a quick and easy way for virtually anyone with access to a computer and a webcam to create a 3D mesh of their own face based on a single 2D frontal image in real-time. This paper is organized as follows: in Section 2 we will discuss related work in this area, in Section 3 we outline the specifics of our algorithm and its components, in Section 4 we discuss the resulting 3D faces our algorithm produces, in Section 5 we conclude our current work, and in Section 6 we discuss our future work with this project, including improvements and future applications.

2 Related Work

As briefly mentioned in Section 1, other solutions have been proposed for generating 3D face models over the years and can be found both in the literature and in industry. This work [3], published in 1996, uses depth-by-shadow to generate 3D models and influenced some

- *Kirtzic's research was sponsored by NSF award 0742477*
- *Daescu's research was partially sponsored by NSF award CNS-1035460*

of our work here. One of the most widely recognized advancements in the field came from the development of the 3D morphable model, developed and published by Blanz and Vetter in 1999 [5]. This model essentially reduced the complex rendering of a 3D human face to two vectors: one for the vertices that comprised the structure of the face, and one for the texture that colored the structure, in a one-to-one correspondence. This idea has been used in other works [6] [7], as well as in forming an important basis for our work. Since the development of this model, many other methods have been suggested for improving upon it, but most still utilize this as their core. Jiang et al [4] [7], for example, propose a method which uses the Blanz and Vetter model, but then expands it to account for variant pose, illumination, and expression. Other related work focuses on facial feature detection and recognition from 2D or 3D images using Principle Component Analysis (PCA) as one method [8].

In industry, a number of more recent solutions have been proposed. However, these systems generally suffer from a number of drawbacks making them inaccessible or undesirable to the average user. EA Sports utilizes a system developed by Digimask, to allow the user to create a 3D reconstruction of their own face and insert it into some of their games, such as Tiger Woods PGA Golf 2009 [9] and Facebreaker [10]. The resulting 3D reconstructions are of a fairly high quality, however, this system requires a rather lengthy and complicated process. Basically, one uses a camera to capture frontal and profile images of themselves, then uploads those images to the EA Sports website, where they are manually converted into a 3D mesh of the user's head. That mesh is then downloaded through the user's online console account (XBox Live, PlayStation Network, etc.) and then loaded into the game. The game is then able to replace the player's avatar's head with the newly created one of their likeness. While this process achieves the required results, it is relatively speaking, rather slow and cumbersome and does not operate in real-time.

Other systems require the use of large software components that make using them undesirable. This is because of the number of components that the user has to download and install on their system as well as the space they require. Furthermore, the size of these components makes them difficult if not impossible for use on portable devices. For example, this system [11] requires several large components, such as the Blender 3D modeler which alone uses 72.9 MB of space.

Other systems, such as Faceware 3.0 [12] offered by Image Metrics, while providing very realistic animated face meshes, still require quite a bit of manual work to be done by the company and utilize proprietary software. More specifically, the user only installs a web portal that allows them to upload their images, then must wait for a conversion process and manual adjustments be done by Image Metrics before finally receiving the finished facial mesh in a proprietary file format.

Many game and graphics engines today are designed with very fast and efficient methods in place to render terrains, as they are usually changing at a fast pace while being highly visible and detailed within a game or simulation environment. Several terrain rendering systems can generate a highly detailed terrain in real-time [13] [14] [6]. Our system takes advantage of the high-speed terrain rendering capability of a game or graphics engine by treating the face as though it is simply a terrain (such as a mountainside) and rendering it.

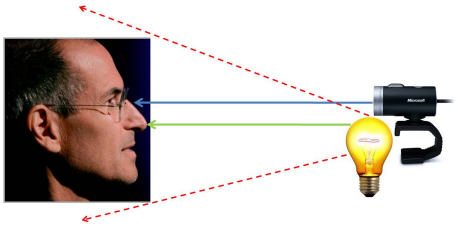

Fig. 1: Camera and light setup for optimal image capture

3 3D Reconstruction Algorithm

Our algorithm has several steps which employ a variety of utilities to achieve the desired result, however at this point some of these steps are being done automatically by the system, while others have to be done manually by the user. As our work continues on this project, we are working to further employ automatic methods to replace the remaining manual ones, as we discuss in Section 5.

3.1 Setup

The accuracy of the system setup is critical for insuring the best possible results. Our system essentially begins with the user being prompted to take a photo of their face while sitting in front of a computer equipped with a webcam. For the algorithm to execute properly, the user needs to use a webcam with a built-in light, or mount a flashlight or other light source next to the camera, with the light pointing directly at the user's face, centered on the nose, as shown in Figure 1.

Also, for optimum results, the room that the user is in should be as dark as possible. The reasons for this will be explained below. For accurate results, the user should have their image take up as much of the screen as possible and take a front-on photo, as shown in Figure 2. The system allows the user to take multiple images until they are satisfied with the image. It should be noted that the hair and ears have been covered for simplicity sake.

It is also worth noting that to demonstrate the accessibility of our system, we implemented it on a standard desktop PC containing commodity hardware and uses open source software. Specifically, we are using a Dell Optiplex GX620 desktop, with a 3.00 GHZ Intel Pentium D CPU, 3GB of RAM, and a NVIDIA 9800 GTX+ GPU with 512 MB of memory. Our system uses the Irrlicht graphics engine version 1.7.2 [15] to render the 3D face,

along with the OpenCV [16] computer vision library version 2.2 to perform all of the computer vision functions.

3.2 Procedure

Once a satisfactory image has been taken the system proceeds in the following manner, which is illustrated in Figure 2. The system then employs a Canny edge detection method on the input image to detect the edges of the face. A Canny edge detector is a multi-step algorithm which employs a Gaussian filter. We chose this because the use of a Gaussian filter to convolve the original image removes noise from the image, making edge detection easier. With this method, we chose to use the L_2 norm (sum of squared differences) as opposed to the L_1 norm (sum of absolute differences). Even though experimentally the L_1 norm performs faster, the L_2 norm typically produces more accurate results. Also, our Canny edge detector method uses a hysteresis procedure with a lower threshold of 10, an upper threshold of 100, and an aperture size of three for it's Sobel filter. The system then employs a face detection algorithm which uses Haars-like features in cascading boosted machine learning to learn to identify a human face over a sample space of 1000 examples. Specifically, the face detection is accomplished by a form of PCA in which the system is trained to recognize eyes, nose, and mouth in the correct orientation. A box is rendered centered around the recognized face, which is overlapped with the edge detection image to identify the correct outer edges of the face. This is determined by identifying the oval that intersects with the box as being the outer face edge. Knowing the coordinates for the box as well as the oval, we then parse each pixel in the image and determine whether it is within the face oval. Thresholding is applied to render anything outside of the face black, eliminating unwanted background images and noise. We determined experimentally that using a threshold of pixel value 30 +/-5 with our thresholding method produces the optimal result.

After the background is subtracted leaving only the face image, the resulting image is then converted into grayscale. This is done using a method that weights the RGB values of a given pixel, then performs a linear combination of these values to determine the pixel's grayscale value. Terrain rendering systems typically use a grayscale height map to determine the elevations of the various terrain features prior to rendering. As far as our system is concerned, the face is simply another terrain with it's corresponding height map (or in this case now, a depth map) used to render the face. By using the lighting technique mentioned in Section 3.1, the depth map derived reflects the depths of the facial features in relation to each other. The result is that the areas of the face that protrude the furthest (and thereby closest to the light source) receive the most light, which results in them being the brightest in the original image. This translates to the highest intensity of illumination in the grayscale image, which then results in their protrusion amount in the resulting depth map, For example, the tip of the nose ends up being the most brightly lit, which translates into the most protruded feature of the face. To enhance this effect, after the grayscale conversion we employ a histogram equalization on the grayscale image, which serves to increase the contrast, thereby drawing a better distinction between the facial features. This technique then can effectively be called a "depth by lighting" method of realizing the degree of protrusion of various facial features. The face is then rotated by 90 degrees about the y-axis, and then by 90 degrees again about the z-axis to achieve the proper orientation for a face.

Although this method on its own does a reasonably good job of representing the depth and contour of a human face, there are still adjustments that need to be made for certain features that the system does not recognize. One obvious problem that arises when you use a simple grayscale conversion of a facial image to determine its depth values is that certain features of the face end up being represented as depressions when they should not be. Specifically, features such as the pupils of the eyes and the eyebrows, as well as other shadows, are usually represented in the grayscale image as being dark in color, and therefore are rendered as being depressions in the face when they are not in real life.

Consequently, an adjustment has to be done to the grayscale image to account for those features by brightening their color intensity to accurately represent their respective protrusions. We are currently doing this adjustment by hand in Adobe Photoshop CS4, where we are using the "airbrush" tool with white "paint" to lighten the areas of the enhanced-histogram image that are still not adequately lightened after the equalization is applied. We are, however, currently working on integrating various methods to automate this process, as discussed in Section 5. The resulting enhanced grayscale image is then rendered as the height map for the face terrain, with the original image then laid over it as a texture.

Blanz and Vetter define their 3D face model with two vectors, S and T, where

$$S = (X_1, Y_1, Z_1, X_2, ..., Y_n, Z_n)^T \in \Re^{3n}$$

and

$$T = (R_1, G_1, B_1, R_2, ..., G_n, B_n)^T \in \Re^{3n}$$

S contains the locations of the vertices that define the facial mesh, represented as X, Y, and Z coordinates in 3D space, where each X_i, Y_i, Z_i denotes the location of vertex i. T contains the color values of the texture at each vertex, represented as RGB color values, where each R_i, G_i, B_i denotes the color value at vertex i. Notice that both vectors are of size n, which indicates that there is a one-to-one correspondence between the structure and texture vectors [5]. Our representation is similar, with the exception that we add a third vector, D, which represents

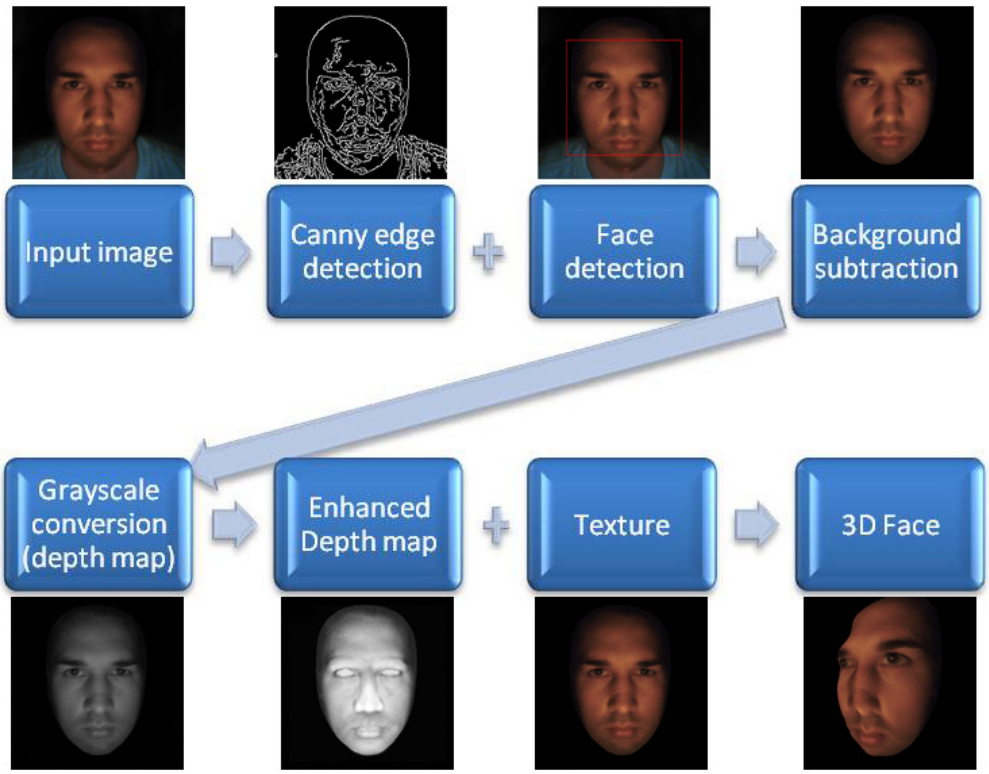

Fig. 2: Summary of algorithm steps

the intensity, or brightness, of each pixel in the enhanced depth map. We define D as

$$D = (I_1, I_2, I_3, ..., I_{m-1}, I_m)$$

were each I_i is a brightness pixel value between 0 and 255. Note also that we now define T and D to be of size m, while S remains of size n. This is due to the fact that S is now defined by the terrain rendering system of the graphics engine, which derives S from the depth map, based upon the resolution defined for the terrain. Thus S no longer maintains a one-to-one correspondence with T and n can be of varying sizes depending upon how the depth map is sampled to generate the terrain structure. However, T does maintain a one-to-one correspondence with D, as D is derived from T without any change in dimension or scale. This allows the original image to be laid on top of the depth map, which guarantees alignment between the two. The resulting face mesh can be moved about like any other mesh and contains collision detection due to its inherent nature as a terrain.

4 Results and Discussion

The current results of this system can be seen in Figure 3 and Figure 4. While far from a perfect rendering, the 3D mesh that our algorithm produces is of remarkably good quality, especially considering the simplicity of the algorithm and small amount of computational power and space that this system requires to perform its task compared to the existing larger systems. By using the basic foundation of the 3D morphable model [5] and keeping the format simple, our 3D face models exist essentially as a vector of vertices, and the corresponding texture, which is simply a matrix of pixel values. This allows our system to be lightweight and portable, and highly accessible, which allows virtually anyone to utilize this system to create 3D facial meshes, quickly, easily, and inexpensively.

Both Irrlicht as well as OpenCV are written in C++, as is our system, so they are all readily available to most programming platforms, from desktops to game consoles. Being as these are both currently available as open source software, we were able to freely strip down these libraries and remove any functions, headers, etc. that weren't directly being used by our system. Our largest component is the Irrlicht graphics engine, which after we stripped it down, weighs in at only 9.85 MB. We are continuing to make our system even more lightweight, so that it can be easily used on portable systems as well as consoles and desktop PCs. For example, even though the size of Apple's iPhone apps can vary greatly, many users seem to prefer apps that are 25MB or less [17], which is what we are currently working to get our system to.

5 Conclusion

We have described our 3D facial reconstruction "Face It" system, which allows the user to generate a 3D face from a single 2D image quickly and easily in real-time. Our

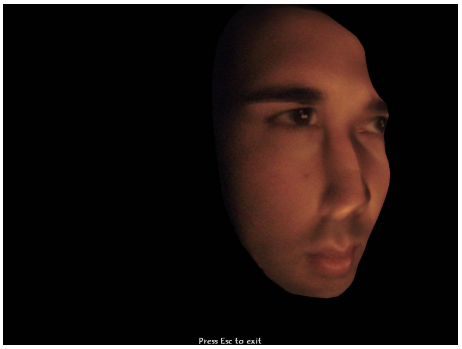

Fig. 3: Algorithm results displaying 1/4 turn

Fig. 4: Algorithm results displaying a profile

system is fast, lightweight, and easily implemented on a wide variety of platforms through its open source components and ubiquitous C++ code. This allows for a wide variety applications in games, simulations, and beyond, on desktop PCs, game consoles, and portable devices. In a world where the lines between virtual worlds and the real world are becoming increasingly blurred, we have come to expect more and more immersion to make our virtual experiences seem as real as possible. Thus, being able to easy insert one's likeness into such virtual worlds quickly and easily helps to make that immersion all the more possible.

6 Future Work

There are several areas where we are currently working to further this project. First is the automation of the remaining manual enhancement processes as identified above. We are currently exploring the use of the the same basic method that we employed for the facial detection, as detailed in Section 2 above, but applying it to individual features. We may then be able to isolate these features in a subwindow of the original image, then apply thresholding to detect the dark region within this subwindow. This region can then be rendered white, giving us an image similar to the one seen in the enhanced grayscale image (Figure 2).

Animation of the face mesh is another area we are working toward. To this end we are developing a generic animated head which can then be morphed to fit a particular face mesh, much in the same way that Blanz and Vetter [5] did with their 3D morphable model. This will ultimately produce a system that is capable of generating an animated 3D mesh of your head and then inserting it into a game or simulation in place of another character's face in real-time, for a wide variety of applications.

Finally, once the system is fully automated and complete, an obvious next step is the implementation of it on a console or smart phone platform. This would, however, require the conversion of the existing system to the other platform's language and structure.

References

[1] M. Levine and Y. Yu, "State-of-the-art of 3D facial reconstruction methods for face recognition based on a single 2D training image per person," *Pattern Recognition Letters*, vol. 30, no. 10, pp. 908–913, 2009.

[2] Y. Luo and M. Gavrilova, "3d facial model synthesis using voronoi approach," in *Voronoi Diagrams in Science and Engineering, 2006. ISVD '06. 3rd International Symposium on*, july 2006, pp. 132 –137.

[3] J. Atick, P. Griffin, and A. Redlich, "Statistical approach to shape from shading: Reconstruction of 3d face surfaces from single 2d images," *Neural Computation*, vol. 8, no. 6, pp. 1321–1340, 1996.

[4] Y. Hu, D. Jiang, S. Yan, L. Zhang, and H. zhang, "Automatic 3d reconstruction for face recognition," in *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, may 2004, pp. 843 – 848.

[5] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3D faces," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 187–194.

[6] L. Hu, P. Sander, and H. Hoppe, "Parallel view-dependent level-of-detail control," *IEEE Transactions on Visualization and Computer Graphics*, pp. 718–728, 2009.

[7] D. Jiang, Y. Hu, S. Yan, L. Zhang, H. Zhang, and W. Gao, "Efficient 3D reconstruction for face recognition," *Pattern Recognition*, vol. 38, no. 6, pp. 787–798, 2005.

[8] Y. Wang, C.-S. Chua, and Y.-K. Ho, "Facial feature detection and face recognition from 2d and 3d images," *Pattern Recognition Letters*, vol. 23, no. 10, pp. 1191 – 1202, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865502000661

[9] EASports. (2011, May) Tiger woods pga tour 09. [Online]. Available: http://www.ea.com/games/tiger-woods-pga-tour-09

[10] ——. (2011, May) Facebreaker. [Online]. Available: http://www.ea.com/games/facebreaker

[11] Q. Zhonghua. (2010) Qu zhonghua's blog. [Online]. Available: http://www.quzhonghua.me/2010/07/simple-workflow-putting-yourself-on-web.html

[12] ImageMetrics. (2011, May) Faceware 3.0. [Online]. Available: http://www.image-metrics.com/index.html

[13] R. A. Apu and M. L. Gavrilova, "Adaptive mesh generation for real-time terrain modeling," in *Proceedings of the twentieth annual symposium on Computational geometry*, ser. SCG '04. New York, NY, USA: ACM, 2004, pp. 447–448. [Online]. Available: http://doi.acm.org/10.1145/997817.997884

[14] H. Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering," *Visualization Conference, IEEE*, vol. 0, p. 35, 1998.

[15] Irrlicht. (2011, May) Irrlicht: Lightening fast realtime 3d engine. [Online]. Available: http://irrlicht.sourceforge.net/

[16] WillowGarage. (2011) Opencv wiki. [Online]. Available: http://opencv.willowgarage.com/wiki/

[17] MacRumors. (2011) The average size of an iphone app. [Online]. Available: http://forums.macrumors.com/showthread.php?t=1133210l

[18] K. Hwang and Z. Xu, *Scalable parallel computing: technology, architecture, programming*. WCB/McGraw-Hill, 1998.