Creating a Face Editor Using Kinect 2.0

Grabovskyi V. A.
Faculty of electronics and computer technologies
Lviv Ivan Franko National University
Lviv, Ukraine
volodymyr.grabovskyi@lnu.edu.ua

Martynovych O.I.
Faculty of electronics and computer technologies
Lviv Ivan Franko National University
Lviv, Ukraine
olehmartynovych13@gmail.com

Abstract—**This paper presents the results of the creation software on Kinect 2.0 basis that is intended to develop a 3D model of the face, simulate various situations and problems that arise during its creation, and consider some tools and methods for their solution. The possibility of correct setting of the original face image to its 3D model and the factors influencing the accuracy of this process are analyzed.**

Index Terms—**Face editor; 3D Face; Kinect 2.0; *Kinect X-Box One;* XAML**

I. INTRODUCTION

Today, face detection is very relevant. In many intelligence systems that use face identification, face recognition algorithms are used for a photo. Face Recognition is currently one of the most popular methods of biometric identification. Facial fixation is now as simple as photographing a person's face.

However, in order for a photo for a 3D face image to be useful for recognition, certain features of the face must be followed. The following requirements must be met: the face must be fixed in the frontal form; the image must not be stretched unevenly; the face should be evenly lit; the face should be in a state of neutral expression with open eyes and closed lips. Photos that match these features will qualify as matching photos that can be registered.

One of the main tasks facing the face recognition system is the complexity of getting good frontal images. The camera should be able to get the full front view of the object. Nevertheless, people have different growths, as well as different forms and features of the face. Therefore, the camera should be able to be positioned at any angle, so that the photo is not perfectly frontal [1].

Matching 3D faces for recognition is a complex task caused by the presence of expression variations, missing data and exhaustive values. A lot of attention is paid to solving this problem, as evidenced by numerous publications in recent years [2-6]. The authors are used different methods and algorithms to create 3D models of the person whose main purpose is to get such a 3D image that is as much as possible consistent with the real person and was convenient at work.

II. METHODS AND TECHNOLOGIES

In this paper are described the technologies used to create the 3D face creation tool – Kinect 2.0 [7, 8], AFogre.NET, OpenCV HaarClassifier, WPF – as well as classes and libraries of the .NET platform for working with photos and media. Even with the help of a large technology stack, creating a realistic 3D model of the face model is not a trivial task.

With the launch of the Kinect X-Box One, developers have the right technology to create a real 3D model of a person without any difficulty. The Kinect 2.0 sensor is a high-tech hardware equipped with a 1920×1080 Full-HD camera, which takes photos of good quality. There is also an infrared volume sensor that can output a 512×424 bulk image. Information from this sensor is probably the best currently available now. It is also possible to capture images from the video. Requirements for video use are fairly standard – 30 frames per second.

However, there are other options for image registration (also at 30 frames per second), such as HD Face, which is one of the most advanced faces tracking library out there. This technology not only does it detect the human face, but it also allows to access over 1,000 facial points in the 3D space in real-time within a few milliseconds

Kinect 2.0 uses a set of standard reference triangles with 2630 triangles. Their use leads to the fact that the model for the face will look really realistic. The indices of triangles that determine all surfaces in the face model are the same for all face models created by Kinect 2.0. The Kinect SDK 2.0 sensor [7] provides the coordinates of the triangles.

Each face in the HD Face has 1347 characteristic points. Kinect 2.0 uses a set of standard reference triangles with 2630 triangles to build a face model. Their use leads to the fact that the model for the face will look really realistic. The indices of triangles that determine all surfaces in the face model are the same for all face models created by Kinect 2.0. The coordinates of the triangles are provided by the Kinect SDK 2.0 sensor [7].

Due to the large variety and unique features of individual faces, it is impossible to simulate the perfect model for each person. With the HD Face API, the ability to choose the most common forms of personality, such as: generic, round, broad, long, oval-wide, oval is implemented. There are defined also 5 key points for each face: the right and left eyes, the tip of the nose, the mouth and the chin. The models differ in their volume, the position of the key points and the distances between them.

A special algorithm for determining the distances between the key points of the image is used to provide that the face image is correctly fitted to the 3D model. The algorithm can be described by the following steps:

1. Find the distance between the eyes of the face image, which is imposed on the 3D model.
2. Find the distance between the eyes on the 3D model of face

image.

3. Find the distance from the tip of the nose to the middle of the eye-connecting line for the image of the face that will overlap on the 3D model.
4. Find the distance from the tip of the nose to the middle of the eye-connecting line for the 3D model of face image.
5. Stretch horizontally the image of the face that will be imposed on the 3D model, based on the coefficient obtained by dividing the distance defined in 1 to a distance defined in 2.
6. Stretch vertically the image of the face that will impose on the 3D model, based on the coefficient obtained by dividing the distance defined in 3 to a distance defined in 4.
7. For a stretched image obtained using application 6, find the vertical distance from the tip of the nose to the upper lip.
8. Find the vertical distance from the tip of the nose to the upper lip for the 3D model of the face.
9. Stretch (or compress) vertically according to the coefficient, obtained by dividing the distance defined in 8 to a distance defined in 7, the image of the face that will imposed on the 3D model.
10. For the image thus obtained, find the vertical distance from the upper lip to the chin.
11. For the image of the face that will be imposed on the 3D model find the vertical distance from the upper lip to the chin.
12. Final step: Stretch the image (or compress it) vertically by the coefficient obtained by dividing the distance defined in 10 to a distance defined in 11.

Now points of the eye will be aligned. However, for some face models, the resulting new image can be significantly deformed. When impose to a 3D model, this image should be stretched or compressed, depending on how the face shape on the overlay image matches the 3D model of the face to which the image is superimposed.

To estimate the quality of the fitting, a method based on "tensile coefficients" is used. The names of the coefficients are determined by the corresponding key points of the image that will overlap on the 3D model of the face. Their using provides stretching of the image on a number equal to the value of this coefficient. For the above-mentioned key points of the face there are four tensile coefficients that are responsible for the distance:

k_1 – from the middle of the right eye to the middle of the left eye;
k_2 – from the tip of the nose to the centre of the eye;
k_3 – from the tip of the nose to the upper lip;
k_4 – from the upper lip to the chin.

The ratio between the coefficients k1 and k_2 is called the "eye-nose" error and is responsible for the area in the image from eye to nose. The coefficient k_3 is called "nose-mouth" error and is responsible for the image area from the nose to the mouth. The coefficient k_4 (the so-called "mouth-chin" error) corresponds to the area of the image from the mouth to the chin.

The values of the corresponding coefficients for the overlay image and the values of the coefficients of the model 3D face image should be approximately equal to each other (if the coefficients are similar the areas represented by this coefficient in the image is well imposed on the face model). The ratios of the corresponding distances (of the coefficients that were found by the above-described algorithm for finding the distances between the key points in the image) for the overlay image and for the 3D model to which this image is superimposed will show how well the image fits around this face model and how much it needs to be stretched or compressed. A result for which the ratio of the coefficients for the overplayed image to the corresponding coefficients of the 3D model will be equal to or close to one will be considered good. If the corresponding value of this ratio is greater than one, then the image of the face that fitted to this 3D model needs to stretch on this value, and if smaller - to compress it.

The correction of these deviations must be made using certain weighting factors. Since the stretching of the image taking into account the eye-nose error occurs throughout the face, the weight factor for it is assigned a higher. On the other hand, stretching the mouth-nose involves stretching the nose down, and stretching the "mouth-chin" involves only stretching down the mouth. Accordingly, he is assigned a weight less than an "eye-nose" error, but more than a "mouth-chin" error. The current weight ratio is 4 for the “eye-nose”, 2 for the “nose-mouth” and 1 for the "chin-mouth". Ideally, for a good fitting face to model, a stretched image of the face should not be deformed.

3D model, created in the HD Face, can be manufactured by .NET Framework classes. For example, "System.Windows.Media.Media3D" using MeshGeometry 3D for simulation, "Viewport3D" – for viewing in the "WPF", "PerspectiveCamera", "AmbientLight" and "DirectionalLight" windows – for visualization of the model under different lighting conditions and viewing angles. "AForge.NET" classes provide filters for processing the original image, which allows changing its brightness and contrast. OpenCV provides "HaarClassifier" to selection facial and eye contours from input images [9].

".NET.Drawing" classes are used to handle GDI+ (Graphics Device Interface) images, one of the three main components that, together with the kernel and the Windows API, are basic for the Windows interface. The "System.Windows.Media" classes are used for presentations in the WPF window (Windows Presentation Foundation – the API for the graphical user interface and part of the .NET Framework) [10].

III. Realization

The project is elaborated on Microsoft .NET Framework 4.5 and is supported by Windows. The logic of the program is implemented using the C # programming language. Microsoft Visual Studio 2017 was used to create the project. To work correctly, you need to install .NET Framework 4.5 or higher. The program interface is implemented using WPF and System.Windows.Media classes.

The main file of the program is MainWindow.xaml, which implements the functionality of the main window of the pro-

gram. Window 1-7.xaml files implements interfaces of the auxiliaries programs for working with images and layouts of faces. Implementation of the program interface is done using the XAML Markup Language. At the beginning, the XAML file identifies the top-level element of the Window – the program window in which the Grid element is defined, which is a top-level container and in which other elements can be added. Each element can have certain attributes with their properties, which allow you to change the size, background, position of the elements of the window. Menu items and their properties describe in <ContextMenu> and <MenuItem>. The Click property specifies the names of event handlers that implement the functionality of the interface elements.

The window of the main interface of the program is shown in Fig. 1

The program has the ability to choose six forms of the face: generic, round, wide, long, oval-wide, oval. The templates for the corresponding 3D faceplates are located on the right panel of the developed interface (1 in Figure 1). Above it are located two switches (2 in Fig. 1), which allow to show or close the grid of face model.

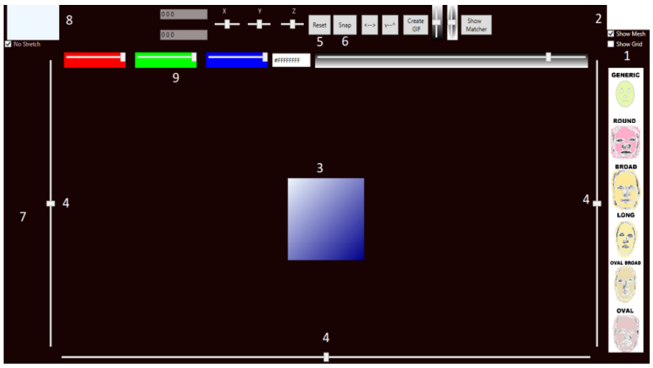

Fig. 1. General view of the program interface:

1 – panel of 3D faces layouts ; 2 – grid face display switches; 3 – cube – the basis for 3D faces layouts; 4 – adjustment sliders of viewing ; 5 – "Reset" button; 6 – "Snap" button ; 7 – panel of face model images; 8 – an element of opening and downloading images; 9 – RGB colour palette.

In the centre of the workspace there is a cube (3 in Fig. 1), which is the basis for the 3D layout of the face and allows to move and rotate the face layout in 3D space. A change in the position of the cube (facial look at different angles) is performed using the view adjustment slider (4 in Fig. 1).

When choosing any of the six shapes of the face in panel 1 in Fig. 1, the layout of the face corresponding to this choice is loaded. To return all the default positions and camera rotation settings, you need to click on the "Reset" button (5 in Fig. 1), or double-click on the rendered face model.

The starting element for fitting is a photo of the face model, which can be saved with the "Snap" button (6 in Fig. 1). It is a 3D face, stored in memory and displayed in the vertical panel on the left (7 in Fig. 1). The last 6 shots are displayed here. You can scroll through the rest of the saved images using the mouse wheel. By clicking on the selected image, you can view it or save it to a file in the Snap project folder.

In the upper left corner (8 in Figure 1) there is an item that allows you to open and download 3D faces. The image can also be downloaded, placed in the centre of the workspace and minimal processed by the RGB colour palette (9 in Figure 1), which also shows the colour code in hexadecimal format and changes in brightness and contrast.

When the face image is selected and loaded, the program will try to find the eyes (Fig. 2). Eye detection is done using the HaarClassifier of OpenCV. The red circular areas on the image are the key points of the face fitting process. Also, a magnifying glass will appear that will show the large picture of the selected area in the face image (1 in Fig. 2).

Fig. 2. Assigning of key points on a face image

1 – magnifying glass; 2 – check box to ensure symmetry of the eyes; 3 – "Best fit" button; 4 – "Update" button.

To impose an image on a face model, you need to bind it to some invariant points on the faceplate. To precisely place key points, you need to select them, and then use the magnifying glass to view the converged image of the selected area and accurately move the key points to the desired part of the image. You can use the "Align eyes" checkbox (2 in Fig. 2) to ensure the symmetry of the eyes. The "Best fit" button (3 in Fig. 2) is used to get the model that is best suited for the new face, and the "Update" button (4 in Fig. 2) – to use the current selected face model. In the process of overlaying an image on a face model, the program interface has the following appearance (Fig. 3).

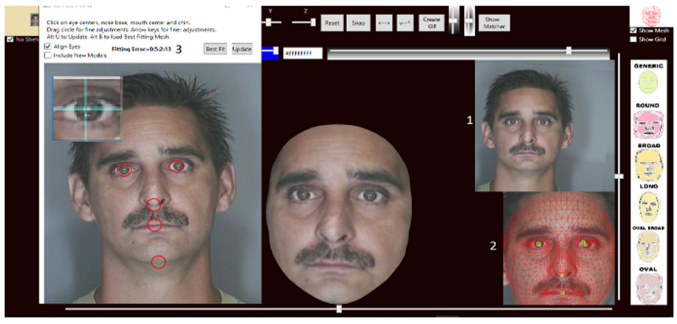

Fig. 3. Face image overlay:

1 – stretched image of the face; 2 – levelled fixation of the stretched face to the model; 3 – "Fitting error".

After updating the model with a new face, one should keep in mind the following:

- The image at the upper right (1 in Fig 3) is a stretched face that will be used as a texture.
- The image at the bottom right (2 in Fig. 3) is a flattened of stretched face fixation to the model.
- "Fitting error" (3 in Fig. 3) shows 4 parameters: <eyes, nose> <nose-mouth> <mouth-chin>, <total error>.

After attaching of the face, you can move and rotate the camera to get the desired look. The "Snap" button saves 3D facial image in memory. After saving the image is available on the left panel (7 in Fig. 1). As a result, the image fitted on the face model, will look like this (Fig. 4):

Fig. 4. Imposed image on the face model

IV. Example of using

One of the main problems in solving the problem of creating a 3D face image is the error in overlays of a face image on a 3D model. The problem is that the face of each person has its own peculiarities concerning the shape of the face and is impossible of perfectly simulating the universal 3D face model that will fit each face image. With the above-described face wrapping algorithm and use of the entered general error of the fitting, you can choose a 3D face model that is best suited for the provided face image.

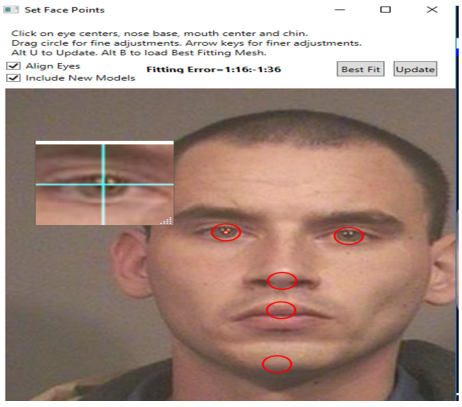

Fig. 5. Configuration of the fitting image to the 3D face model

Fitting error compensation will help to adjust the position of the facial points. Let's consider the process of reducing it on the example (Fig. 5). In this case, the "eye-nose" error is -1, the "nose-mouth" error is 16, the "mouth-chin" error is -1 and the total error (calculated with the appointment of weights: 4 for the "eye-nose", 2 for "nose-mouth" and 1 for "mouth-chin") - 36. In order to compensate for the negative "eye-nose" error, you need to bring the key points of the eyes closer together or lower the key point of the nose. To compensate for the positive "eye-nose" error, you must move the key points of the eye further apart or move the key point of the nose above. To compensate for the negative "nose-mouth" error, you need to move the key nose and mouth points apart. To reduce the positive value of this error, you need to move these key points closer together. To compensate the negative "mouth-chin" error, you need to increase the distance between the corresponding key points in the image, and the positive one - to reduce the distance between them.

Correct moving the key points of the image to compensate for the errors of fitting it to the 3D model will allow to get an image of the face with proportions close to the original one.

V. Conclusion

On the basis of Kinect 2.0 a software has been created that analyzes the possibilities of creating a three-dimensional face model, simulated the situations and problems that arise during this process, as well as examines some tools and solutions for them solving. Special errors rates are used to increase the correct installation of the initial image to the 3D layout of the face. Correctly using the errors values when fitting a face image to the 3D model and compensating for their size, we can choose the optimal 3D face model for a particular person's image. However for perfectly fitting of an image to its 3D model, a unique 3D face model is required for each individual.

References

[1] "Kinect for Windows SDK 2.0. Getting started" [Electronic source]. – Available from:https://msdn.microsoft.com/en-us/library/hh835354.aspx

[2] D. Smeets, P. Claes, D. Vandermeulen, J. G. Clement. "Objective 3D face recognition: Evolution, approaches and challenges", Forensic Science International, 2010, 201, pp. 125–132.

[3] D. Smeetsa, J. Keustermansa, D. Vandermeulena, P. Suetensa. "meshSIFT: Local Surface Features for 3D Face Recognition under Expression Variations and Partial ,Computer Vision and Image Understanding, 2013, 117, Is. 2, pp. 158-169.

[4] N. Smolyanskiy, .C. Huitema, L. Liang, S. E. Anderson "Real-time 3D face tracking based on active appearance model constrained by depth data", Image and Vision Computing, 2014, 32, pp. 860–869.

[5] G. Hu, Fei Yan, J. Kittler, W. Christmas, Chi Ho Chan, Z. Feng, P. Huber. "Efficient 3D morphable face model fitting", Pattern Recognition, 2017, 67, pp. 366-379.

[6] Luo Jiang, Juyong Zhang, Bailin Deng, Hao Li, Ligang Liu. "3D Face Reconstruction with Geometry Details from a Single Image", Accepted by IEEE Transactions on Image Processing, 2018.

[7] M. Rahman."Beginning Microsoft Kinect for Windows SDK 2.0 : Motion and Depth Sensing for Natural Interfaces", Apress, 2017.

[8] "Kinect for Windows Software Development Kit" [Electronic source]. – Available from: https://www.techspot.com/drivers/driver/file/

[9] "WPF overview" [Electronic source]. – Avaible from: https://www.tutorialspoint.com/wpf/wpf_overview.htm

[10] A.Trojelsen. "Pro C# 5.0 and the .NET Framework 4.5", Sixth Edition, Apress, 2012.