Eurographics Italian Chapter Conference (2011)
A. F. Abate, M. Nappi, and G. Tortora (Editors)

3D-ize U! A Real-time 3D Head-model Texture Generator for Android

Stefano Boi[1], Fabio Sorrentino[1], Stefano Marras[1,2], and Riccardo Scateni[1]

[1]University of Cagliari, Dept. of Mathematics and Computer Science - Italy
[2]Visual Computing Laboratory, ISTI-CNR, Pisa - Italy

Abstract

Recently, the number of applications developed for smartphones has dramatically increased; however, at the moment, applications having the purpose of creating and displaying 3D models are quite rare. The goal of this work is to build an application that allows the user to see the virtual three-dimensional representations of their friends and interact with them. The main challenge is to achieve results similar to those that a computer would produce, optimizing the process to deal with the constraints due to the technology used. Since there are no similar mobile applications, this work will make possible to create a base onto which will be possible to realize applications that have customized 3D models as a common feature.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Image processing—Warping
Keywords:Android, 3D Head models, Real-time texturing, Photo

1. Introduction

According to [Mar10], the economic value of the mobile market will increase from 6,8 billions dollar in 2010 to 25 billions dollar in 2015. This statistic also states that the cause of the increment is the widespread diffusion of the smartphones, along with the progressive lowering of connection fees and the increment of use of mobile applications.
Lots of applications that allow the user to *play* with reality, entertaining him in a way considered unthinkable till a few time ago, were born in the last years. Giving a quick look to the applications market for mobile system, it's easy to notice that the photo-editing oriented applications are getting downloaded more and more.
The idea of building a tool for *playing* with the face of the people, based on the use of a 3D model, comes up from this fact. The application has been developed using the Android platform, a choice mainly due to the availability of low price devices. When developing an application for mobile, one has to take into account the peculiarities of mobile phones hardware, such as the processor speed and the memory capacity, that have a way lower limit than standard computers. For more information about the Android platform and the application development, see [com11c], [com11b] and [com11a]
The result of this work is an application named **3D-ize U!** that allows the user to build and display a 3D model of one of his friends simply using a picture. In order to reach the goal, the application goes through several steps:
- **Taking** a picture, or choosing an existing one;
- **Detecting** a set of *feature points* in the picture;
- **Processing** the image using warping techniques in order to create a texture;
- **Applying** the texture onto a predefined 3D model;
- **Rendering** the 3D model and his texture;
- **Interacting** with the model;

This paper is organized as follows: in section 2 we will introduce the ideas behind this work, section 3 describes the phases of the application, then performance are examined in section 5.1; section 5.2 deals with the limitations of the algorithms; finally section 5.3 discusses about future works.

2. Background

Warping is the set of proceedings focused on the transformation and deformation of shapes. This technique has been

DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2011/041-046

(a) (b) (c)

Figure 1: Final results

largely used in shape animation, modeling, analysis and matching (see for example [VFG*09] [MN06]). Basically, the image is deformed in order to distort the geometry of the objects depicted in the scene; it's a process mainly used to correct the distortion caused by the lenses, but has been recently used also for artistic photo editing. Using the warping technique, it is possible to modify an image aiming to align a set of image features to the same features on a different image, usually referred as the *template* image. It's also possible to calculate the amount of transformation needed to align the features: in this way it is possible to choose among a pool of images picking the one that is most similar to the template, to recognize people or objects that appear in damaged or distorted images, even if the object itself is disguised.
Intuitively, a graphical object U consists of any processable entity in a computer graphics system: points, lines, surfaces, images and so on. In the particular case where the graphical object is an image I, the transformation $T : I \rightarrow R^n$ defined only for the points of I is named *intrinsic* transformation.

The easiest way to work on an image consists of defining a global transformation of I, that is, a transformation applied uniformly onto every pixel of the image; however, it's really hard to transform a specific object of the scene by means of a global image transformation. Instead, this goal can be achieved working locally on a subset of pixels. A good strategy to build local transformations is based on the idea of subdividing the object into smaller pieces, then defining a transformation for each piece separately and finally merging all the pieces together to obtain a new image. Working locally has also the advantage of approximating each transformation using simple functions, obtaining good results and reducing the computational cost. Triangles and quadrilaterals has been widely used in computer graphics as blocks to build graphical objects. Dealing with a triangulated image or a subdivision made of a quadrilateral mesh is a rather diffused scenario and the techniques to compute the transformation from a polygon to another gain considerable importance (see [BKP*10]).
One of the challenges of the techniques based on locally defined transformations is the need of computing new attribute values for every point of the deformed object using only the values store in the original one.

Considering the particular case in which has been calculated the affine transformation between two triangles ABC and DEF, the transformation T on the vertices is defined as $T(A) = D$, $T(B) = E$ and $T(C) = F$. So it is possible to apply T to every point inside the triangle using barycentric coordinates.

A point p of a triangle ABC has barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$, that is

$$p = \lambda_1 A + \lambda_2 B + \lambda_3 C$$

when $\lambda_i \geq 0$ and $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

The transformation to obtain the position of point p in relation to the new reference triangle is the following:

$$\begin{aligned} q &= T(p) \\ &= T(\lambda_1 A + \lambda_2 B + \lambda_3 C) \\ &= \lambda_1 T(A) + \lambda_2 T(B) + \lambda_3 T(C) \\ &= \lambda_1 D + \lambda_2 E + \lambda_3 F \end{aligned}$$

After studying and analyzing the mathematical model, the application development chapter 3.3 will explain how these techniques can be implemented in order to obtain the desired deformation.

3. Development phases

The project has been divided into different phases: image acquisition (3.1, detection of feature points (3.2), image warping (3.3), texturing/rendering of the model (3.4) and finally user interaction (3.5).

3.1. Image acquisition

The first step is the image acquisition. The user must provide an image to the application, and he can do this in two ways: by taking a picture "on-the-fly", or using an existing photo.
Once that the picture has been selected, a special module verifies the goodness of the image and notifies the users of the necessity to take the picture again (or choose a better one).
In the specific case, a good image is an image that has good contrast, good lighting, a well-defined background and where the face of the person portrayed is clearly visible, making possible a full-automatic face detection step. To identify the face, the Android *face detection* feature has been used; thanks to this feature, it is possible to correctly locate the face, and center it into an ellipse that will be then used to cut the image. For best results it is necessary that the user modifies the ellipse to contain the face. Through the use of finger gestures, the user can zoom in or out at will; once the size of the oval is adjusted it's possible to move to the next phase, the detection of facial feature points.

3.2. Feature points

The feature points are the key element in the process. A good tradeoff in terms of quality and quantity needs to be achieved, because on the one hand too few points produce unreliable results and poor accuracy, while on the other hand too many points make the application computationally heavier and less practical to use. As seen in [SB08], the most important points to be identified are represented by the eyes, the nose and the mouth. For best results, however, the eyebrows and the silhouette of the face should also be used. Let's analyze every feature point used in the process (see Figure 2):

- 4 points for the left eye;
- 2 points for the left eyebrow;
- 4 points for the right eye;
- 2 points for the left eyebrow;
- 3 points for the nose;
- 4 points for the mouth;
- 12 points for the face silhouette;

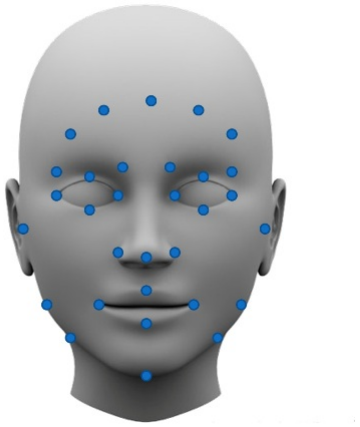
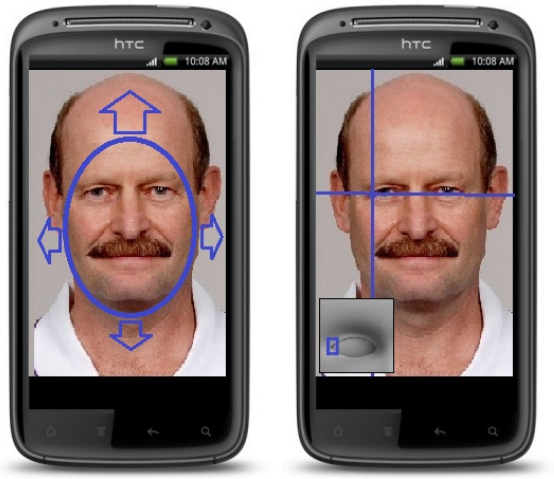


Figure 2: All feature points taken with the user interaction.

The 12 points of the face silhouette are automatically extracted from the ellipse placed over the image by the user.
The acquisition of the remaining points is performed in a sequential manner, using a specific panel that tells the user which part of the face he has to pick. The application shows a cursor made of by a vertical and a horizontal line intersecting the point touched. The user can scroll with the fingers in order to refine the selection and maximize the accuracy.
Usually, the texture contains all the information to represent the entire surface of the 3D model; in this case the model is a head, so the texture should include information about the front, the sides and the back. In this work the choice has been to acquire a single frontal picture, and as a result of this, the resulting texture is incomplete. This problem will be addressed and solved in section 3.4.

For each point the user has to press a dedicated button to confirm the coordinates indicated, that saves the current point and displays the image of the next point to be identified.



(a) (b)

Figure 3: Oval fitting activity, point selection activity

This assisted acquisition allows to create a point-to-point correspondence map between the acquired picture and the final texture, and this will be particularly useful for the warping phase . At the end of this procedure every point is known and the process of warping that will generate the texture to apply to the 3D model will start.

3.3. Image processing/warping

After the acquisition of the feature points, the image has to be warped in order to fit a pre-existing template image. The goal of this step is to modify the original image in order to use it for texturing the three dimensional model of the head. Thanks to a 3D model deep analysis, the application knows both the coordinates of the texels associated with the feature points and the correspondences with the points of the 3D model. The information is then used to build the template that will serve as reference for the warping.
Since the system can't detect the position of feature points in an automatic way, the detection of this points is an user duty. As soon as the user locates the points, it becomes possible to warp the original picture in order to fit on the template. The warping operation is performed using the concept of barycentric coordinates ([Pic06]). The first step of this process consists in carrying out the triangulation of the feature points indicated by the user. Since the position of the feature points in the acquired image is known, and also the corresponding template points are known, it is possible to triangulate the image to obtain its discrete subdivision, working locally on the pixels. The triangulation is a classic *Delaunay triangulation* computed incrementally (imple-

mentation provided in [Che07]). This technique makes possible to incrementally build the triangulation while the user picks the feature points; in this way, both triangulation and selection of feature points end at the same time.

First, the algorithm builds a triangular bounding box such that every point of the triangulation, named as set P, falls within it.

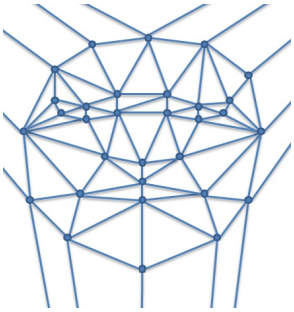


Figure 4: Delauney triangulation of feature points

For every point belonging to P, it is necessary to search for the triangle where it lies. This triangle is then split into three new triangles. This operation is trivial for the first point, when the only existing triangle is the bounding box. Subsequently, a test is performed over the newly created edges in order to avoid the presence of illegal edges. In case of illegal edges, the structure is updated to solve the problem. This is the incremental way to build the Delaunay triangulation of the feature points; the triangles are stored keeping track of the correspondences between the original triangles and the new ones.

The algorithm performs only one triangulation using the set of feature points defined by the user, then build a hash map containing the correspondences between those points and the ones located on the template image. In this way, for each triangle T of the image triangulation, exists a triangle T' that has three vertices corresponding to the 3 feature points of T. The only difference is that these points have different spatial coordinates. To overcome this problem, it is possible to represent every pixel of the image by means of barycentric coordinates, with the assumption of knowing which triangle contains the specified pixel. Then, every pixel is processed locally, moving it from the old reference system (the triangle that contains it) to the new reference system (the corresponding triangle with moved vertices).

Once identified the new location of the pixel it is possible to assign it his old color. Repeating this process for each pixel leads to the desired transformation. However, the image may require some other adjustment to avoid that some pixels may appear incorrectly. The reason is that, because of the discrete nature of pixels, by undergoing this transformation not every pixel of the final texture happens to be the destination of a pixel of the original image. Applying a smoothing filter that, for every not colored pixel, gives him the average intensity of the 8-neighbor pixels, overcomes this problem. Each pixel of the neighborhood is taken into account to calculate the mean if and only if his intensity is not zero.

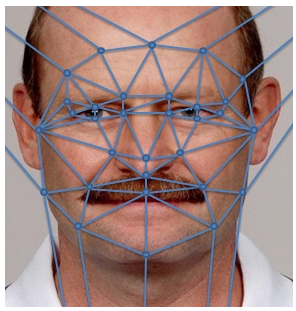


Figure 5: Delauney triangulation overlying the texture

The choice of this technique has been made after finding out that achieving the image deformation by means of a global transformation was too time expensive, and result could not be achieved in a reasonable amount of time.

Several tests are in fact been made with tools such as MatLab, using cubic and bi-cubic interpolation, obtaining very good results; however, even if a good computer could compute these functions quite fast, the problem turned out to be too heavy to be carried on by a mobile device. Obtaining the texture from a single picture has the drawback that there is not enough information to texture the sides and the back of the head. In this work, the developers preferred to exclude the details of hair, which would require to be independently studied and that can be integrated in the application afterwards.

For this reason, the model obtained has no hair and the choice to assign the color to the skin in the back of the head has been achieved using some kind of averaging of the available information.

Working on the morphology of the human face, it is possible to state that except in some areas such as eyes, nose and mouth, the skin has the same regular pattern everywhere, excluding the area on the jaw where, especially in men, there may be a thick hair that hides the natural color of the skin. To complete the texture, gaining a realistic effect, the information on the pattern of the skin is taken directly from cheeks and forehead. To improve this result and reduce the difference between the color of the facial skin and the synthetic skin, the color of the facial image is modified, applying a different enlightenment, similar to the one of the pattern just computed. In this way the color difference is almost not noticeable. The texture is then ready to be adapted to the model.

3.4. Texturing and rendering

The Android operative system provides a subset of the OpenGL graphics library to render 3D objects. The only thing needed is a parser that read and build the model, while the visualization part is carried by the os. It' obvious to notice that the more complex the model, the more slow the visualization. See [PAM*07] for further details on OpenGL for mobile system.
Another constraint to consider is the number of vertices and faces that is representable by an array of short integers, limiting the complexity of the models displayable. Besides that, the number of vertices and faces has been limited to 1016 vertices and 1999 faces due to the excessive computational cost, and also because on small screens is not necessary to have highly detailed objects to get a good result. In the testing phase the model has been directly inserted in the application package as a RAW file, then the file is parsed to retrieve the coordinates of the vertices, information about faces, vertex normals and the texture map. Notice that it is possible to save the texture map on a file because those information never change going from a texture to another. The model is now ready for the visualization; after the model is rendered, the user can finally interact with it.

3.5. User interaction

As said before, the aim of this work is to create an application that would allow a number of future developments having in common the use of custom 3D models and real time texturing using photos taken from the mobile device. Currently, the user can interact with the model like any 3D viewer: he can zoom, rotate and scale the displayed object. Furthermore, having different models with different facial expressions, it will be possible to replicate different expressions depending on user interaction.

4. Performance

Having some useful / funny / cool features is not enough to make a "good" application. A good application needs to perform these operations quickly. During several tests the time spent by various processes has always been monitored, to identify the steps in which the code was computationally too expensive.

In detail, the application has been tested on a device with internal processor Qualcomm MSM7201A 528 Mhz and 192 Mb RAM. Since phase 1 and 2 are strictly depending from the user, they aren't taken into account, while phase 3 requires in average 53 seconds and phase 4 requires 43 seconds. Times are hardware dependent; brand new smartphones have 1.5Ghz dual core processors, and these operations can be performed in much less time. Table 1 provides timings with respect to different mobile devices. An interesting way to speedup the process is the use of a remote server for executing the third phase and fourth phase; this alternative will be better discussed in following chapter.

5. Conclusions

5.1. Results

The results of the work reflects the objectives set at the beginning of the development phase: developing an application for the growing mobile world, based on the Android platform, that allows the user to have a real time textured 3D model, where the texture can be obtained just taking a picture of someone's face. Actually the application can:

- take a picture, or choose it from the sd card;
- recognize the presence of a face in the picture and draw a silhouette around it; eventually, the user can manually adjust it;
- help the user to identify the feature points, and pick them;
- warp the image so that it can be used as a texture for the 3D model;
- show the 3D model and let the user interact with it through touchscreen, as a classic 3D viewer.

5.2. Limitations

The points 1,2 and 3 are tightly linked and the navigation between screens is very fluid; on the other hand, since the application is currently in a test phase, the points 4 and 5 are executed as stand alone applications, but can be easily integrated with the others modules. When the testing phase, particularly long on Android because of the availability of several different devices, will be over, the above-mentioned points will be well connected and the application will be released, making possible for others developers to integrate new modules. During this work some problems has risen, mainly due to the devices used.
Another problem solved during the development concerns the warping of the image. This deformation is necessary for our work to obtain a texture easily applicable to the 3D model, but the warping is based on mathematical functions that can be very complex, and the tradeoff between complexity and quality result is not fair. Despite the fact that the results are truly better using complex functions than using simple ones, the quality difference doesn't justify the gap in computation time. Hence, on mobile platforms, linear functions must be used in order to have the output back in a reasonable amount of time; in particular, the transformation used is based on the idea of barycentric coordinates, therefore a linear transformation.
Despite of the decision of working with linear functions and although the triangulation algorithm has been optimized using a trapezoidal map, since the device used for the testing phase is a first generation Android mobile phone it became impossible to use high resolution images in the warping phase. This device happened to be around 970 times slower than a mid-level laptop computer.

Table 1: Table reporting the timing for the execution of step 3 of the algorithm.

Device	Release Year	Approx. Time (sec)
HTC Magic	2009	56
Samsung Galaxy S	2010	30
Samsung Galaxy S2	2011	2
HTC Sensation	2011	2

Some little problems, due to the Android platform, rose during the phase of the acquisition of the image. When the image is acquired, the device takes some time to store it either in the internal memory or in the SD card, delaying any work on that image until the storing process is finished.

5.3. Future works

The purpose of this work is merely to build a base onto which it will be possible to create similar applications. The use of 3D customized models offers several ideas to develop in the near future.

For example, it would be interesting to make an application where people could try different hairstyles or hair color, or see themselves with beard or mustache. It would also be possible to use the images of a future father and mother to have an idea of the face of their son, or use the images of a father and his son to highlight the common features and see how the grown up child will appear, or warp the faces to see how our friends would be if they were fat, thin, elder or younger.

An even more interesting application could use other 3D models, for example glasses, applying them on the existing head. Glasses, earrings, hats or any other head-related accessory could help to make an interesting application. From a ludic point of view, it could be fun to texture different 3D models like animal heads or even common items with the image of its own friends. In the end, it could be even possible to create a community to share the contents; the future developments are lots and very different from each other.

References

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LÉVY B.: *Polygon Mesh Processing*. A. K. Peters Ltd., 2010. 2

[Che07] CHEW L. P.: Voronoi/delauanay applet, 2007. www.cs.cornell.edu/home/chew/chew.html. 4

[com11a] Android lifecycle and working, January - August 2011. http://developer.android.com. 1

[com11b] Android story and evolution, January - August 2011. www.androidiani.com. 1

[com11c] Android tutorials, January - August 2011. www.anddev.org. 1

[Mar10] MARKETS M. .: World mobile applications market - advanced technologies, global forecast, August 2010. www.marketsandmarkets.com. 1

[MN06] MULLENS S., NOTLEY S.: Image morphing. www.stephenmullens.co.uk. 2

[PAM*07] PULLI K., AARNIO T., MIETTINEN V., ROIMELA K., VAARALA J.: *Mobile 3D Graphics: with OpenGL ES and M3G*. Morgan Kaufmann Publishers, Inc., November 2007. 5

[Pic06] PICKOVER C. A.: *The Mobius Strip*. Thunder's Mouth Press, 2006. 3

[SB08] SOHAIL A. S. M., BHATTACHARYA P.: Detection of facial feature points using anthropometric face model. In *Signal Processing for Image Enhancement and Multimedia Processing*, vol. 31 of *Multimedia Systems and Applications Series*. Springer US, 2008, pp. 189–200. 3

[VFG*09] VELHO L., FRERY A., GOMES J., VELHO L., FRERY A., GOMES J.: Warping and morphing. In *Image Processing for Computer Graphics and Vision*, Texts in Computer Science. Springer London, 2009, pp. 387–412. 2