CS597 Computer Vision

Sylvain Vignaud

12th May 2002

Abstract

This paper presents a fast technique for creating 3D model of a face from two pictures of a face, one from the front, the other from the side.

Such technique can have many applications, ranging from user interface improvement to video-conferencing.

This method is based on a general 3D mesh that will be deformed and textured. Another possible method would be to use image warping between the two pictures, but this limits too much the range of results (point of view, model deformation...).

We will first discuss how features points can be find, how the model can be distorted and then how the texture mapping occurs.

Contents

Chapter 1

Detecting feature points

There are three ways of detecting features:

- the first method is based on knowledge about the feature texture
- the second method is based on color segmentation
- manual feature points location by the user

In this project, I chose to use manual feature points location for the sake of simplicity and accuracy. Indeed automatic feature points detection is quite an intricate problem and it can introduce noise in the location of the features, which would degrade the quality of the result.

1.1 Texture-based method

We must compute for each feature point a set of possible correlation matrix. Such a matrix multiplied by a sub-part of the picture will give a maximum result when the sub-image is precisely located on the feature point.

The inconvenient is that in order to be able to takes into account all the possible shapes and size of the feature inside the pictures, we have to define not only a single correlation matrix per feature, but a set of possible correlation matrix.

For example, in order to detect a corner of an eye, up to 9 correlation matrices are commonly used, in order to take into account three positions of the eyelid and three sizes of the eye.

1.2 Color-based method

This method segments the picture into area that are colored in similar ways. For example, the whole mouth would be filtered and colored in red, the skin would have a single color on the whole face, and no shading would be noticeable.

The image must be filtered so that for a given pixel, every neighboring pixels are colored in the same way is the color difference does not exceed a threshold.

This method does not introduce as much computation as the previous method. It also will not introduce as much noise as the previous method in most of the cases.

Still some inaccuracy can be introduced by the shading on the face. There are also problem in filtering picture of a character having facial hair.

1.3 Naming the features

Using some geometry constraint on the model (the eyes are almost aligned, the nose is centered between and under the eyes, the mouth is centered and behind the nose...) we can match each possible feature point that have been previously detected with the feature points that are required.

Chapter 2

Altering the shape of the 3D model

2.1 Computing a deform vector for each feature point

We have find the 2D coordinates of every feature point in the two pictures. We also know where the feature points are located on the 3D model.

In the front view, four points define the general size of the face: we can use the top, bottom, left and right point to know the width/height ratio of the head and apply it to our 3D model. After this the four previously named point are well located. The same technique can be used with the side view to alter the depth of the 3D model.

The next step is to place all the remaining features. For every single feature, we know its location in the two pictures. Therefore we can have their coordinates according to the four previously placed features in the front picture, having the top and bottom points defining the Y axis, and the left and right points defining the X axis. We can then use these 2D coordinates to compute the predicted position of the 3D feature in the (X,Y) plane. By using the same technique on the side view, we can also compute the Z coordinate of every feature points.

Knowing the original 3D location and the new 3D location of each feature point, we know how each feature must be moved.

The final step is to compute such displacement vector for every remaining vertexes of the 3D mesh.

2.2 Altering a vertex depending on the feature points

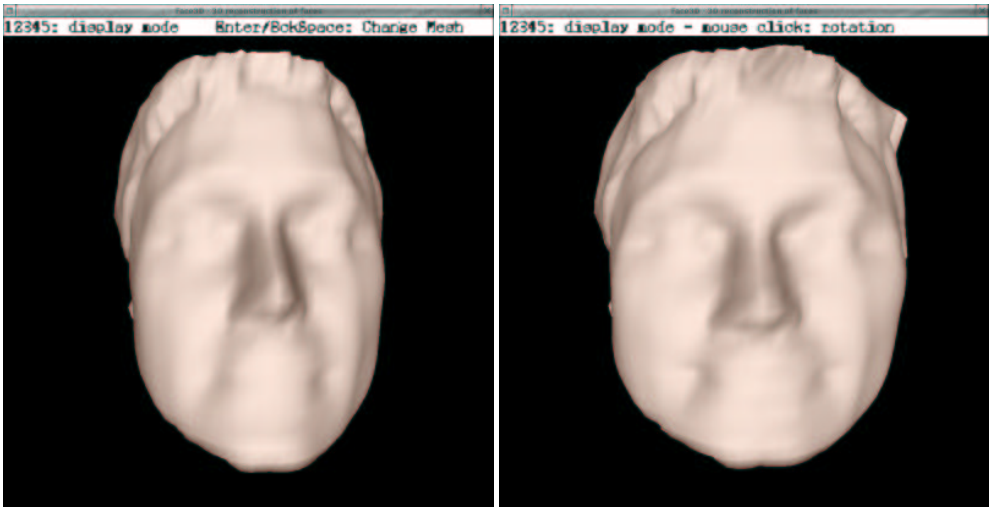
A movement applied to a feature point alter the 3D model only locally: when one point is moved, the whole face is not altered, only the neighboring vertexes should be moved.

We can assume that the movement of the neighboring vertexes is inversely proportional to the square of the distance between this vertex and the feature point.

Therefore we can compute the displacement vector of every single vertex as begin a weighted sum of all the feature point displacements, where the weight is the inverse of the square of the distance.

This last step gives us the 3D mesh after alteration. We now have a 3D mesh that represent the head of the character, but still its rendering would not be very good-looking as we do not know how to render each triangle of the mesh: the only rendering method we can use is to display the model with some lighting.

That's why, in order to achieve more realistic results, we have to use the two input picture to texture-map the mesh.

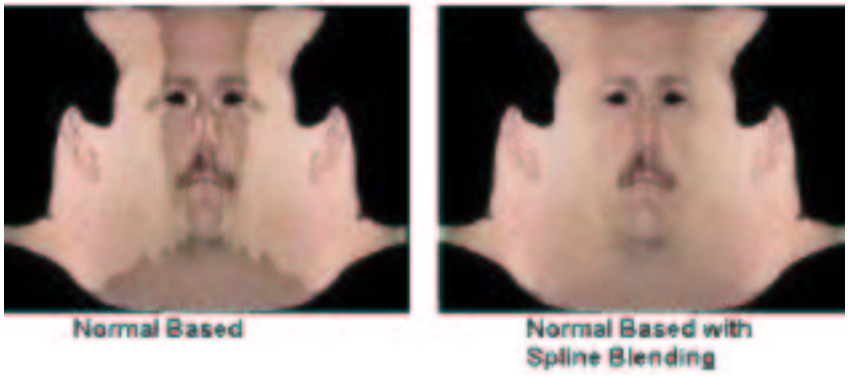


Chapter 3

Mapping textures on the 3D model

There is two main way to texture-map the model:

- compute a unique texture and use spherical mapping:



- use both pictures as two textures and use some blending between them.

I am using the second method with some OpenGL blending, as it gives good results.

3.1 Computation of the texture coordinates

We can assume that the camera was properly placed, that is the camera was placed and looking on an axis that is perpendicular to the picture.

Therefore, assuming the face does not takes the whole field of view of the camera, we can say that the rays from the camera to the face are almost parallel, and we can use a planar projection model.

Therefore, using the front picture as our first texture, we can use the 2D coordinates of the vertexes into the pictures as our texture coordinates. We already know the position of the feature points, so the position of the other vertexes can be computed using linear interpolation, as we previously did to alter the 3D mesh.

The same algorithm with the second picture will give us the second set of texture coordinates.

Now we have two textures with two sets of texture coordinates. The final step will be to blend them together.

3.2 Computation of the texture blending

As we are using OpenGL for the rendering process, we will use one of its feature to blend the two textures together:

- we first display the mesh with the first texture (R,G,B,Alpha).
- we set the blending operation of OpenGL to be glBlendFunc(GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA) where Alpha comes from the first texture.
- we display the mesh with the second texture and blending switched on.

Therefor we have to compute Alpha for each pixel of the first texture, that is for each pixel of the front picture.

We know that using the first texture will give better results on the front of the 3D mesh, on the other hand using the second texture will give better results on the side of the mesh.

So we have to say to OpenGL to use mainly the first texture on the front of the model (Alpha=0) and mostly the second texture on the sides of the mesh (Alpha=255).

To compute Alpha, we will take into account the X position of the pixel according to the left, right and center of the face in the picture.

Two thresholds are defined. Let's d be the distance between the distance on the X axis between the pixel and the center of the face, with d=0 in the center of the face and d=1 on the side limits of the face:

- if d is smaller than the first threshold, only the front texture will be used.
- if d is greater than the second threshold, only the second texture will be used.
- Else we have to blend both texture. I use a non linear interpolation between 0% and 100%. I use $Alpha = \left(\frac{d-LIMIT1}{LIMIT2-LIMIT1}\right)^2$, giving $0 \leq Alpha \leq 1$.

Now that the mesh has been reshaped, and textured, we are ready to see some results of this technique.

Chapter 4

Results

Due to the face that one of the person whose face has been pictured does not want his face to appear in this document, I will present results only on two head.

4.1 Test on the first character

With these input:

These results are produced:

Of course the glasses are not rendered very well: As they are projected in the same way as the head on the images, they are rendered as if they were part of the skin. The easier way to handle this would be to take pictures of the user without glasses, and after to add a 3D mesh of the glasses.

The hair does not looks that good.

We can also note on the last screenshot that there is a white area in the hair: this comes from the fact that the hair-style is not symmetric, whereas we suppose it is when we use the same side texture for both the left and the right of the face: this hole comes from the front picture which includes the asymmetric hair-style.

Still the user is very recognizable. Both front view and side view of the user look quite good.

4.2 Test on the second character

With these inputs:

These results are produced:

Y

Even while the mouth position has changed between the two pictures, the generated mesh looks good.

Chapter 5

Conclusion

In this paper we have shown how to create a 3D mesh of a character's head thanks to two pictures of his head and a 3D model.

This program takes only a fraction of one second to do the required computation on a middle-class computer (Duron 800MHz under Linux), so it could be easily used in a commercial program.

Bibliography

[1] Yoshimitsu AOKI and Shuji HASHIMOTO, Adaptive Head Modeling System and Its Application

[2] Eric Cosatto and Hans Peter Grag, Photo-Realistic Talking Heads from Image Samples

[3] A Murat Tekalp and Jörn Ostermann, Face and 2D Mesh Animation in MPEG-4

[4] Stéphane Valente and Jean-Luc Dugelay, Face Tracking and Realistic Animations for Telecommunicant Clones

Appendix A

Test launch procedure

There are three sample sets.
Each set is composed of:

- two pictures
- one text file that contains the feature locations in both image and on the 3D mesh.

To launch one of the test, you can type:

- make t1
- make t2
- make t3

This command will compile the program when required, and will launch it with the proper test pictures and feature points.

To see the sources, open the joined tar.gz archive that contains the source files, the Makefile plus the test samples.

Appendix B

Program user interface

Right now the program is not that user friendly. There is some help displayed at the top of the 3D view - or in the text terminal while displaying the pictures.

The following keys have functionality:

- 1: display the front picture. In this mode, feature points can be placed.
- 2: display the side picture. In this mode, feature points can be placed.
- 3: display the wire-frame 3D view. In this mode, feature points can be placed.
- 4: display the filled 3D view with lighting.
- 5: display the filled 3D view with texture and lighting.
- Enter: Alter the shape of the model to match the pictures
  (only in mode 4 and 5).
- Enter: start placing features
  (only available in mode 1, 2 or 3).
- Space: save the position of the current feature and go to the next feature
  (only in mode 1, 2 or 3).
- Backspace: discard the position of the current feature and go to the previous feature
  (only in mode 1, 2 or 3).
- Arrows: move the 3D model on the X and Y axis
  (only in mode 3, 4 or 5).
- PgUp/.PgDown: move the 3D model on the Z axis
  (only in mode 3, 4 or 5).

- Escape, q: quit the program.
- Mouse, left button: place a feature point (only in mode 1, 2 or 3).
- Mouse, left button: rotate the 3D mesh (only in mode 4 or 5).
- Mouse, right button: rotate the 3D mesh (only in mode 3, 4 or 5).