Assignment Description

Due: Thu Aug 18, 23:59:59 IST

For this assignment we will build a morph effect. This algorithm was used (and probably developed) for the morphing of faces at the end of Michael Jackson's classic "Black or White" video (click to see from 5m30s in). Your simplest result will be the following:

See morphF.gif

Assignment Instructions

Your assignment is to implement the research paper "Feature-Based Image Metamorphosis" by T. Beier and S. Neely from SIGGRAPH 1992. This assignment combines 2D geometry (feature vectors describing warps) and image processing (blending two images and warping pixels). Your main references will be the skeleton code and the research paper.

Start by reading Feature-Based Image Metamorphosis, available here (html) and here (pdf). Pay special attention to the whole of Section 3, which describes the algorithm in detail. **COMPLETELY READ THE PAPER BEFORE ATTEMPTING ANY IMPLEMENTATION!** Many things are explained in footnotes or later on in the paper, so you will avoid getting stuck by reading the whole paper first.

(There is a small typo in the paper: line 7 of the pseudocode should be D[i]=X'[i]-X, not X'[i]-X[i].)

We will provide two example morphs for you: warping an F (above), and morphing Bush into Obama (below). We expect you to submit an image sequence of 11 images, morphing between the first and second image for both of these examples in t=0.1 increments from 0.0 to 1.0.

The paper defines three parameters: a, b and p. We set these parameters as { a : 0.5, b : 1, p : 0.2 } since that gave us decent results. Feel free to experiment with these values, and note the parameters you used in a Readme.txt file.

You can clamp the edges of the image -- thus, requesting pixels beyond the border of the image gives you the horizontally (or vertically) closest pixel in the image. See the documentation of the sampleBilinear() function for more details.

Download the skeleton code here. (Aug 10, 11pm)

Code: You need to edit the files **morph.cpp** and **LineSegment.hpp**. Please do not change any other file. Every function that you must edit (and that we will test) is marked with a "TODO" comment. Specifically, the functions are:

- LineSegment.hpp: lineParameter, signedLineDistance, segmentDistance
- morph.cpp: sampleBilinear, distortImage, blendImages, morphImages

Look at the comments in the files. They will help you write the functions correctly according to spec. Please test the functions individually before putting it all together. **It is ok to add additional functions if you need them but don't change the signatures of the above functions at all.**

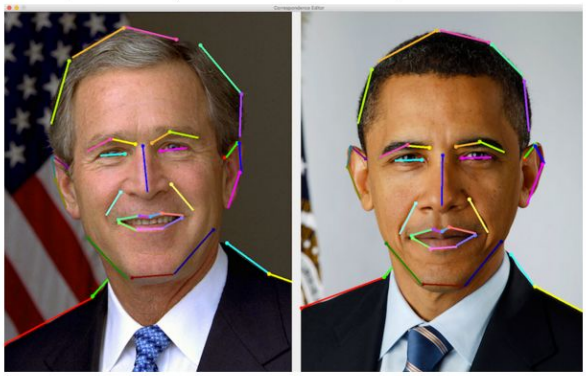
Data: Inside the "data" folder of the skeleton code zip you will find the following input files:

- 1a.jpg - Bush-Obama first source image
- 1b.jpg - Bush-Obama second source image
- 1p.txt - Bush-Obama feature segments file

- 2a.jpg - F-morph first source image
- 2b.jpg - F-morph second source image
- 2p.txt - F-morph feature segments file

Correspondence Editor

You can use the Java-based Correspondence Editor program (in the "editor" subdirectory of the skeleton zip) to view, draw and save feature segments. Once you load the program up with two images, draw segments by clicking and dragging with the mouse. You must first draw a feature for the left image, and then the corresponding one for the right image. If you load an existing set of features, newly drawn ones will be appended to this set. We use the same input format as this editor produces. For example the default Bush-Obama feature segments look like this:



Assignment Hints

Implementing this algorithm consists of five distinctly different milestones:

- Writing the geometric functions of LineSegment
- Blending two images
- Bilinear sampling of colors from an image
- Distort an image using only one feature vector
- Taking multiple feature vectors into account to distort an image

We suggest that you follow the above order. First complete LineSegment. Then, implement the blending of two images, and use linear interpolation for each channel of each of the pixels. **Do not assume images will always have 4 channels, use the numChannels() function to get the actual number of channels**. Please see chapter 2.10 of Shirley (Fundamentals of Computer Graphics) for a better handling of linear interpolation of vectors than Wikipedia.

Once your blending works, turn blending off, and write the code for bilinear sampling (make sure to sanity check it with various inputs!), followed by the simple algorithm to warp an image with a single feature vector. Write to disk the result of the first and second images warped, and confirm that this looks right. For the F, if you only use the second feature vector, you will rotate both images in opposite directions.

Once you can distort an image using a single feature vector, now extend the algorithm to use all the feature vectors. Check the individual results of this before putting it together with the blend and rendering an image sequence.

At all times, saving images from individual steps -- `image.save("debug.png")` -- is a great way to test individual components. Test code units in isolation, with known input/output pairs.

Understanding the time (t) parameter: Time is used to describe both how "far into" the morph we are, or how "far along" a linear interpolation process is. Time is normalized to the range 0 to 1. This means that at time 0, you see the first source image (for example, 1a.jpg) during the morph, and at time 1 you see the second source image (for example, 1b.jpg). Similarly, if you linearly interpolate two colors, at t=0 the resulting color is the first color, and at t=1 the resulting color is the second color. At t=0.5 both colors and both images in the morph are equally mixed -- that is, a blend of 50% color 1 and 50% color 2.

Using Algebra3.hpp for vectors: In this assignment you will be working with vectors, for which code is provided in Algebra3.hpp. Operations such as minus and dot product of Vec2's are implemented in this library, so you can use that for all your mathematical calculations. Look at the top of Algebra3.hpp to see what is available.

Extra Credit

You can get up to 10% of extra credit on this project. Create your own morphs by selecting two images of your choice and building the feature segments and parameters for it. You may submit at most 3 custom morphs.

Submission Instructions

You need to submit both your results and source code. Your submission directory should be called a2, and it should contain:

- Readme.txt (add any comments here)
- LineSegment.hpp
- morph.cpp
- A "results" folder, containing
    - F0.0.png, F0.1.png, F0.2.png, ..., F1.0.png - the results of morphing the F
    - BushObama0.0.png, BushObama0.1.png, ..., BushObama1.0.png - the results of morphing Bush into Obama
    - **Up to 3 other sequences**, named similarly, for your extra credit submissions. You should also include the **original source and target images, and the set of feature correspondences**. If the image sequence is FooBar0.0.png, FooBar0.1.png..., then the source image should be FooBarA.png, the target image FooBarB.png, and the correspondence file should be called FooBarP.txt. Finally, you must document the **corresponding values of a, b and p** (if you used anything other than the default) in your Readme.txt, along with a note of each such extra credit entry.

Please do NOT include any other files! We will drop your two source files into our build system to verify your results.

You should **zip** this folder (do **NOT** use .tar, .tar.gz, .rar, .7z or any other format) and submit it on Moodle. If you log in to Moodle, you should find an assignment called "A2: Feature-Based Image Metamorphosis" set up. You can use "Add submission" to upload your zip file. You may resubmit any number of times before the deadline.

Expected results

Here is our rendering of Bush into Obama:

See morphBO_small.gif