Assignment P1 : MORPHING

Computer Graphics 2

Due 9 February 1995 (before midnight)

In this assignment you will produce a "morph" animation of your face into another student's face. Each of you will generate two seconds of animation, which we will record onto video for you. The final video will thus be a metamorphosis through each of the faces in the class.

Unless you feel adventurous and want to develop a morphing method of your own[1], the algorithm that you should implement is described in the paper: Thaddeus Beier and Shawn Neely, "Feature-Based Image Metamorphosis", *SIGGRAPH '92 Proceedings*, pp. 35–42, which we have provided. Read it.

A morph is a simultaneous warp of the image shape and a cross-dissolve of the image colors. The cross-dissolve is the easy part; controlling and doing the warp is the hard part. The warp is controlled by defining a correspondence between the two pictures. The correspondence should map eyes to eyes, mouth to mouth, chin to chin, ears to ears, etc., to get the smoothest transformations possible. A variety of methods for specifying a correspondence between two images is possible; Beier and Neely use pairs of line segments to define the correspondence (read their paper for the full details). Their method is simple and easy to control relative to most other methods.

The class directory, which we will abbreviate *classdir* henceforth, is `/afs/andrew.cmu.edu/scs/cs/15-463`. Your directory for this assignment is *classdir*`/students/`*your_last_name*`/p1`, and we call this *p1dir*.

We will give you a starting image ("picture A") and an ending image ("picture B") for your animation sequence. If you had your picture taken by us then picture A will be your face. Picture B will be another student or other primate. You will find these pictures in *p1dir*`/a.ppm` and *p1dir*`/b.ppm`[2].

You'll morph still picture A into still picture B and produce 61 frames of animation numbered 0-60, where frame 0 must be identical to picture A and frame 60 must be identical to picture B. In the video, each frame will be displayed for 1/30 of a second.

Your job will consist of two parts: writing a program to perform the morph given the correspondence, and using an interactive program provided by us to define the correspondence.

Part 1: Write morph. Write a program called `morph` which, when run with the arguments:

`morph` *picA picB linesfile warpfrac dissolvefrac morphpic*

[1] Which you're welcome to do, but in this case much of this document is irrelevant to you. See the note on *Other Morph Methods* at the end.

[2] Until these files are on line, around 1/30/95, experiment with whatever you can find in *classdir*`/pub/p1/pix`.

will read picture files *picA* and *picB*, line segment file *linesfile*, and warp between A and B by *warpfrac*, and cross-dissolve by *dissolvefrac*, writing the resulting picture to picture file *morphpic*.

We will use PPM picture file format (see `rtfm ppm` for documentation).[3] Use the binary ("raw") variant of the format to save disk space. You will need to write (or find) routines to read and write PPM picture file format, but this should be easy, since the PPM format is so simple (ascii header followed by pixels in scanline order, with 3 bytes per pixel).

The line segment file *linesfile* contains a list of line segment pairs for pictures A and B, plus three parameters at the beginning of the file. Our morph editor will write out this file, but your code will need to read the file. The format is ascii:

$a\ b\ p\ n$
$P_{1Ax}\ P_{1Ay}\ Q_{1Ax}\ Q_{1Ay}\ P_{1Bx}\ P_{1By}\ Q_{1Bx}\ Q_{1By}$
$P_{2Ax}\ P_{2Ay}\ Q_{2Ax}\ Q_{2Ay}\ P_{2Bx}\ P_{2By}\ Q_{2Bx}\ Q_{2By}$
...
$P_{nAx}\ P_{nAy}\ Q_{nAx}\ Q_{nAy}\ P_{nBx}\ P_{nBy}\ Q_{nBx}\ Q_{nBy}$

where n is the number of line segment pairs, and the parameters a, b, and p are explained in the paper. The ith line segment pair consists of the line segment in picture A between points (P_{iAx}, P_{iAy}) and (Q_{iAx}, Q_{iAy}), and the line segment in picture B between points (P_{iBx}, P_{iBy}) and (Q_{iBx}, Q_{iBy}). Coordinates in the file are normalized to the range [0,1], so you will need to multiply the x's by the width of the pictures and multiply the y's by the height of the pictures.

For example, the following lines file contains two line segment pairs, and defines a correspondence involving a 90 degree rotation:

```
.01 2 .2 2
0 0 1 0 1 0 1 1
0 0 0 1 1 0 0 0
```

This is a short example. A typical lines file to specify a good morph might require 40 line segment pairs or so. See figures 7-11 of the paper.

The parameters *warpfrac* and *dissolvefrac* control warping and cross-dissolve, respectively. They both lie in the range [0,1]. They are the only parameters that will vary from frame to frame in the animation. For your starting frame, they will both equal 0, and for your ending frame, they will both equal 1. When you produce the final frame files, you will want them to be equal for all frames, but during debugging of your morph program, it can be helpful to have independent control of these two parameters.

Your `morph` program will need to perform the following (conceptual) steps:

1. Read in two picture files and one lines file.

2. Compute "destination" line segments by linearly interpolating between PQ_{iA} and PQ_{iB} by *warpfraction*. These line segments define the "destination shape".

3. Warp picture A to the destination shape, computing a new picture. We'll call the result "Warped A".

[3] On Andrew, the manual page can be found in `/usr/contributed/man/man5/ppm.5`.

4. Warp picture B to the destination shape, computing a new picture. We'll call the result "Warped B".

5. Cross dissolve between Warped A and Warped B by *dissolvefrac*.

6. Write the resulting picture to a picture file.

In the warping steps (3 & 4), do bilinear interpolation, as described in lecture, so that the resampled picture won't appear blocky. Note that steps 3-6 can be combined into one pass; the list above is a list of *conceptual* steps.

The *morphpic* that you write out should have the same size in pixels as *picA* and *picB*.

Part 2: Design your piece of the animation. The second part of the assignment involves designing your piece of the animation by running our `medit` program to specify the correspondence between pictures A and B.

`medit` is an X Windows program that lets you interactively create, move, and delete line segments, and at the push of a button it will run your `morph` program (by forking off a process) so you can preview still frames from your animation. Sliders in `medit` allow you to control the morph fractions. To run `medit`, give it the name of your two picture files: "`medit` *picA picB*". After editing the line segments to your satisfaction, hit the `Save Lines File` button to save them to a file. Use the filename `morph.lines` for the line segments that you use for your final animation. For more documentation on `medit`, see *classdir*`/pub/p1/doc/medit.doc`.

The parameters a, b, and p affecting the correspondence are described in the paper. If you find that the picture is not going where you tell it (e.g. the eyes aren't lined up, even though you've placed line segments on them), it could be because a is too large, b is too small, or p is too large. We recommend the default settings: $a = .01$, $b = 2$, and $p = .2$.

If your `morph` program is not working, or you want to compare its results to that of a "correct" morph program, choose `Algorithm: Their Morph` button and the official morph program will be used.

You can generate frames by setting up your `morph` and `morph.lines` files and running the shell script *classdir*`/pub/p1/shell/genanim`, which writes out picture files into your *p1dir*`/anim` directory. You can preview these frames using the `xplay` program, which we also provide. `xplay` reads a sequence of PPM picture files and displays them in quick succession. Run "`xplay -`" for more info. You can find `medit` and `xplay` in *classdir*`/pub/p1/bin`.

Note that generating your final frame files could take many hours of CPU time, so LEAVE YOURSELF A DAY OR TWO BEFORE THE DEADLINE TO COMPUTE YOUR FRAMES. These files will be big, so be aware of your 30 MB quota.

What to Submit. A correct submission consists of five parts that should go in *p1dir*.

1. Source code to your `morph` program.

2. A working executable of your `morph` program that we will be able to run (executable on either a DEC, SGI, or Sun4).

3. 61 frames with filenames `anim/f00.ppm`, `anim/f01.ppm`, ..., `anim/f60.ppm`. These frames must be generated by *your* `morph` program (we will check this).

4. `morph.lines`, the line segment file from which you generated your frames.

5. A file named `README` containing a brief summary of the contents of your source files (a line or two each should do) and any comments to us that you'd like to make about problems you had or extra things you did, if any. Please also tell us how long your morph program takes to run per frame, and what machine type you ran on (be specific, e.g. "DEC 5100").

If your submission is late, it will be penalized by 20% per CMU class day, and your animation might not make it into the videotape. Submissions might also be penalized if the file names do not conform to the standards above.

Tips. Running `morph` at full resolution will probably take 10 seconds to an hour, depending on the speed of your hardware and software and the number of line segments you use. For faster interaction when running the `medit` program, we **strongly** suggest that you work with lower resolution picture files. You could use the program `/usr/contributed/bin/pnmscale` to make shrunk versions of your *picA* and *picB*, then run `medit` on these until your correspondence is well set, only using the full resolution pictures to generate your final frames. If your program is running too slowly, use profiling tools to identify hot spots, pre-compute common subexpressions, try integer arithmetic where possible, and compile with the optimizer.

If you find the quantization objectionable, try the `-newcm` option, which will give the created windows a new colormap and use all 256 colors on an 8-bit display. If your pictures look dark, try the option `-gamma 1.5` to both `medit` and `xplay`.

Other Morph Methods. If you develop your own morph method in place of Beier and Neely's, please check that the warps and animation look smooth. Submit everything in the list above except, in place of item 4, please include any data files that are involved and write up a brief description of how your algorithm works.

Have fun!