Problem Set 3
CMSC 427
Assigned March 27, 2007, Due April 12, 2007

Introduction: This problem set will focus on writing a program to morph one image into another. You will implement the approach described in: "Feature-Based Image Metamorphosis," by T. Beier and S. Neely, *Computer Graphics*, 26(2) 25-42 (available on-line at: http://www.cs.princeton.edu/courses/archive/fall00/cs426/papers/beier92.pdf).

You will be given some basic code, as well as an executable for finished code. Some things to notice about this code:

- BMPImage.cpp and .h define an image class for bmp type images. This includes functions to read in an image and to save an image
- texture.cpp and .h contain functions that load an image from a file and turn them into a texture.
- main.cpp contains code that reads in two images, and provides an interactive tool that allows you to specify corresponding line segments in the two images. These will be needed to create correspondences for morphing. To use this interface, try the following operations:
    - Click left on an image to start a line segment. Drag the mouse to specify the end of the segment.
    - 'u' undoes the last line segment on the left image. 'v' does this for the right image.

You will modify this code to use these line segments to morph between the images. In addition to these features, the executable contains a couple of other features:

- 'm' causes the left image to morph into the right, using the line segments that you've specified in each image. Note that the order and direction used to create the line segments matters. The first line in the left image matches the first one on the right, and the first points of each segment match.
- 'f' causes the left image to fade into the right.
- 's' causes all the images in the morph to be saved in a directory called 'Results' (you should create this directory ahead of time.
- 'r' replays the morphing using the images saved in 'Results'. This is useful if you are morphing larger images, which my program will not morph in real time. Replaying gives you a better sense of the morph.

1. **Warm up (10 points):** As a warm-up exercise, modify your code from problem set 2 to read in some images of your choice and texture map them onto the rectanguloids and/or ground-plane in the scene.
2. **Fade (15 points):** Implement a fade operation. This is begun by typing 'f'. The image on the left of the screen should gradually fade out, being replaced by the image on the right. Let's call the image on the left I, and the image on the right J. If the fade lasts n units of time, then after k units of time pixel (x,y) will have intensities given by $(1-k/n)*I(x,y) + (k/n)*J(x,y)$. You might try, for example, generating a fade that lasts 4-6 seconds, at 30 frames a second.

3. **Morphing (40 points):** In this part, you will implement morphing. This problem will be satisfied by a complete morphing program, but I'm going to suggest that you break this into parts, doing first (a), then modifying this to satisfy (b), and then completing (c). You can receive partial credit for turning in a program that implements (a) or (b).
    a. **Interpolate Lines (10 points):** When the user types 'm' morph lines drawn in the left image into the lines drawn on the right. That is, as a line morphs from the left to the right the positions of its end points are linearly interpolated from their start position to their end position. When you finish this part, the behavior of your program should be the same as in the executable in terms of how the lines move, although the underlying image needn't change.
    b. **Morph the left image (10 points):** Now change the underlying image so that the left image is morphed as the lines move. The interpolated lines from part (a) are used to define coordinates in the new image you generate, and pixel intensities are transferred from the left image to this image. This is like full morphing except that the image you generate isn't a combination of the left and right images, but just the left image. So, for example, with proper placement of lines, you can cause the left image to be compressed, or distorted.
    c. **Morphing (20 points):** Now implement the whole morphing program of Beier and Neely. This means applying your code from part (b) to both the left and right images, and then displaying an interpolation of the two, similar to that produced with fading.
4. **Generate image or video (5 points):** Use your code or mine to generate an interesting or compelling image or video (which can be displayed with 'r'). If you want to use your own code, you should implement 's' and 'r'. A prize will be given to any notable results. **(5 points extra credit for implementing 's' and 'r', up to 5 points extra credit for creative images).**
5. **Extra Credit:** Integrate the results of morphing into your game. For example, create a cube with a texture-mapped image that morphs into something else when you get close to it.
6. **Extra Credit:** In the morphing program, to fill in a pixel in the new image, we compute values (x,y) and use these to get the value of a pixel in the old image. Typically, x and y will not be integers. You can handle this by rounding off x and y. For extra credit, implement subpixel interpolation. So, for example, if we want to get a pixel value at (6.2, 7.3) we should compute this value as a weighted average of the pixel values at (6,7), (6,8), (7,7), (7,8). If you implement this, document clearly where in your code it is done.

A maximum of 20 extra credit points are available.

Paper and Pencil Exercises:

1. **Texture Mapping (10 points)**: The following problems should be done by hand, with written explanations for the results.
    1. Suppose A is a 16x16 matrix such that A(x,y) = x+y. That is, A(0,0) = 0, A(3,7) = 10, ….
    2. Suppose we render some textures, using the commands:
        - `glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 16, 16, 0, GL_RGB, GL_UNSIGNED_BYTE, texImage.getData());`
        - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);`
        - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);`

    I'm going to ask questions about the values that the image pixels will take under different circumstances. For the purposes of this problem, you can assume that the texture coordinate (0,0) corresponds to the first corner of the first pixel in the texture, and that (1,1) corresponds to the last corner of the last pixel. However, if you are unsure about some details of Open GL you can make up anything reasonable, as long as you explain what you are doing.

    a. Suppose we render this texture into an image region that is a 16x16 square of pixels, using the commands:

```
glBegin(GL_QUADS);
      glTexCoord2f(0,0);
      glVertex2f(0, 0);
      glTexCoord2f(1,0);
      glVertex2f(1, 0);
      glTexCoord2f(1,1);
      glVertex2f(1, 1);
      glTexCoord2f(0,1);
      glVertex2f(0,1);
glEnd();
```

    What is the intensity at the image location with coordinates (.2,.2)? At (.2,.6)?

    b. Just for this question, suppose in the command `glTexParameteri` we use GL_LINEAR instead of GL_NEAREST. What is the intensity at the image location with coordinates (.1, .3)?

c. Suppose we render this texture into an image region that is a 16x16 square of pixels, using the commands:

```
glBegin(GL_QUADS);
      glTexCoord2f(0,0);
      glVertex2f(0, 0);
      glTexCoord2f(1,0);
      glVertex2f(.5, 0);
      glTexCoord2f(1,1);
      glVertex2f(.5, 1);
      glTexCoord2f(0,1);
      glVertex2f(0,1);
glEnd();
```

What is the intensity at the image location with coordinates (.2,.2)? At (.2,.6)?

d. Suppose we render this texture into an image region that is a 16x16 square of pixels, using the commands:

```
      glBegin(GL_TRIANGLES);
            glTexCoord2f(.2,.2);
            glVertex2f(.7,0);
            glTexCoord2f(.6,.6);
            glVertex2f(.7,1);
            glTexCoord2f(1,.2);
            glVertex2f(.9,0);
      glEnd();
```

What is the intensity at the image location with coordinates (.8,.2)?

2. **Cubic interpolation (10 points):** In 1D we can interpolate between two pixel values using the equation: $1 - 3d^2 + 2|d|^3$. That is, suppose v is a vector of intensities, and we want to interpolate between v(i) and v(i+1) to figure out the intensity at some intermediate point, v(k), where k is a non-integer $i < k < i+1$. We can interpolate a value for k using cubic interpolation. To do this, let d denote k-i. Then we can set:

$v(k) = v(i)*(1 - 3d^2 + 2|d|^3) + v(i+1)* (1 - 3(1-d)^2 + 2|(1-d)|^3)$

$= v(i)*(1 - 3(k-i)^2 + 2|k-i|^3) + v(i+1)* (1 - 3(i+1-k)^2 + 2|(i+1-k)|^3$

Prove the following properties of cubic interpolation.

   a. It creates a v that is continuous. That is, if j converges to k, then v(j) will converge to v(k). Hint: you only really need to worry about what happens when k is an integer.
   b. It produces a weighted average. That is, v(k) is equal to a weighted combination of v(i) and v(i+1), and these weights always add up to 1.
   c. It is symmetric. That is, if the distance from k to i is d, then v(k) is a weighted sum of v(i) and v(i+1). If instead, the distance from k to i was (1-d), we would reverse the weights we would use when combining v(i) and v(i+1). Another way to look at this, is that if we exchange the values of v(i) and v(i+1), and use (1-d) instead of d, we would like to get the same interpolated value.

d. It is smooth. That is, it has a derivative that is well-defined everywhere. To do this, you mainly need to show that at an integer value, if dk is some tiny delta that is going to zero, then: $v(k) - v(k-dk) = v(k+dk) - v(k)$.

3. Color (10 points)

a. If a pixel has RGB values of (.7, .3, .2), how would you describe its HSV values?
b. If a pixel has a hue of green, a saturation of .5, and a value of .4, describe it with RGB values.
c. Suppose you had a monitor that emitted light that was either cyan, magenta or yellow. How could you use this to create white light?
d. Suppose you have three colors, A, B, and C, represented in RGB by (.7, .4, .3), (.4, .2, .2), and (1,0,0) respectively. Give the RGB representation of a new color that has the hue of A, the saturation of B, and the value of C.