Problem 1 - Displaying topography.

Copy the file lab8prob1data from the class web directory. This is a byte image representing topography on the Big Island of Hawaii, and each increment in pixel value corresponds to 20 meters in height. The image is 735 pixels by 927 lines.

i) Display the elevation data as a grayscale image. Identify the major mountain peaks and main saddle points. Submit on paper with the major features labeled.

ii) Create shaded relief displays of the same data. Use illumination from the top, left, and halfway between the top and left directions. Display and submit.

iii) Create a color version of the map with the following properties, display, and submit. For each pixel in the output image, generate data according to an indexed color table with each entry defined as follows:

   (1) Let the brightness at each point be the intensity in one of the shaded relief images from part (ii), scaled to the range 0-1.

   (2) Let the hue (which can range from 0-1 in matlab) at each point be defined as

   \[ h(x,y) = \frac{255-e(x,y)}{510} \]

   where \( e(x,y) \) is the relative elevation of the pixel scaled to be in the range 0-255. In other words, just use the pixel value from the elevation image.

   (3) Let the saturation (which can range from 0-1 in matlab) at each point be

   \[ s(x,y) = \frac{255-e(x,y)}{400} \]

   Define each pixel according to the above HSB definitions, load into a 3-D array in hsv format, and convert into corresponding RGB values using the matlab command hsv2rgb(). Then load the data into a 3-D array and display and submit.

Problem 2 - Contour maps.

Using the same data as in problem 1, create a contour map using the matlab command contour(). Set the contour intervals at 200 meters. Which peak is highest?

Problem 3 - Perspective views. (This is the first step to generating a flyby such as you might produce in your project animation.)
Copy the input data file lab8prob3data from the class web directory. This is also a map of Hawaii in byte form at 20m/level, but the file is now 512 by 512 pixels.

i) Display the image and its shaded relief version to make sure you have read it in correctly. Submit.

ii) Create a perspective view of this image using one of the sets of equations given in the notes. Use values of height and distance to create a pleasing perspective. Use the shaded relief image to get the intensity of the function that is displayed at each point, and the elevation data to generate the location of the points. Submit.

iii) Generate a rotated view by calling the matlab routine imrotate() on both the elevation and shaded relief data before projecting as in step (ii). Pick an 'interesting' viewing angle, display, and submit.

Extra points: Make the image from (iii) color by using the method of problem 1, part (iii).

Problem 4. Morphing example

Copy the files lab8prob4data1 and lab8prob4data2 from the class web area. These are byte images of portraits of two US presidents. The images are 180 pixels by 256 lines.

i) Display the data as grayscale images. Identify good features to serve as tie points in the morph.

ii) Record the coordinates of each tie point in each image. The matlab command ginput allows you to read the location of the cursor when you click the mouse. From these coordinates, calculate the displacement vectors from one image to the other.

iii) Compute a matrix for x displacements and a matrix for y displacements for each location in the 180 by 256 image. Use the matlab commands meshgrid and griddata, as in the following sample code. Note: you will have to use the correct parameters in the calls for this problem.

```matlab
% generate resampling grid
x=linspace(1,180,180);
y=linspace(1,256,256);
[xi,yi]=meshgrid(x,y);

% shiftx is an array of the shift in x at each point
shiftx=griddata(xdata,ydata,zdata,xi,yi,'linear',{'QJ'});
```

In the above, xdata, ydata, and zdata are the location and shift in x of the tie points, while xi and yi define the x and y coordinates of the output array. You will have to do this for both x and y shifts.
iv) Display the shift matrices and verify that they look correct.

v) Now, implement the morphing code as we discussed in class, which functionally looks something like this:

for k=1:11,
frac=(k-1)/10

% get the new locations of the pixels, moving the first image towards the second, and % the second towards the first
locx = round(xi+shiftx*frac);
locy = round(yi+shifty*frac);
locxx = round(xi-shiftx*(1-frac));
locyy = round(yi-shifty*(1-frac));

% make sure all pixels stay within the array (use the clip.m code in the class directory)
locx=clip(locx,1,180);
locy=clip(locy,1,256);
locxx=clip(locxx,1,180);
locyy=clip(locyy,1,256);

% now map the pixels to their new positions and blend
% here is a simple loop to do it - Warning: use only one method!!
for i=1:180,
for j=1:256,
final(j,i)=w(locy(j,i),locx(j,i))*(1-frac) + l(locyy(j,i),locxx(j,i))*frac;
end;
end;

% alternatively, you can speed things up if you use the following matlab code that % takes advantage of built-in matlab functions
final= w((locx(:)-1)*256+locy(:))*(1-frac) + l((locxx(:)-1)*256+locyy(:))*frac;
final=reshape(final,256,180);

image(final);   % but scale it properly
M(k) = getframe;

end;

Now you can display your movie.

Finally, recompute the movie using only the first image without the blend so that you can see the effects of the warp alone.