CS1115 Fall 2013 Project 5 Due Wednesday November 13 at 11pm

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group. Each problem is worth 10 points. One point may be deducted for poor style. No partners are allowed for the Challenge problem.

Objectives

In this assignment is about image processing and text file manipulations. You will deepen your ability to manipulate 2-dimensional arrays and 3-dimensional arrays that represent images. Read Chapter 12 in *Insight*. You will also gain experience working with online datasets, cell arrays, and structures.

1 Postcards

Download the jpeg files `CornellSnow.jpg`, `CornellBear.jpg`, and `SunBlue.jpg`. The codes you write for this assignment should assume that these files are in the current working directory. This assignment involves the manipulation of 3D arrays. Not every "shortcut" with 3D arrays is legal. And because these arrays are composed of `uint8` values, you have to be very careful with type. Appreciate every line in the following script before you start design your solutions.

```
A = imread('CornellSnow','jpg');  % A is an example of a 3d uint8 color array
C = A;                            % C is a copy of A
C(:,:,1) = 100;                   % Every entry in the first layer has value 100

v = [.7 .2 .4];                   % A sample rgb vector
% Give pixel (20,30) the color specified by v...
C(20,30,1) = uint8(255*v(1));
C(20,30,2) = uint8(255*v(2));
C(20,30,3) = uint8(255*v(3));     % NO: C(20,30,:) = uint8(255*v);

% Create an rgb vector w associated with the color of pixel (3,4)...
r = double(C(3,4,1))/255;
g = double(C(3,4,2))/255;
b = double(C(3,4,3))/255;
w = [r g b];                      % NO: w = double(C(3,4,:))/255;
```

1.1 Borders

Here is the image represented in `CornellSnow.jpg`.

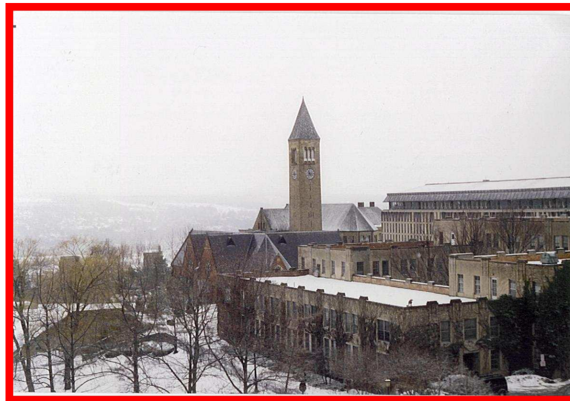
Complete the implementation of

```
function B = AddBorder(A,w,c)
% A is an nRows-by-nCols-by-3  uint8 color array that corresponds to some image I
% w is a real number that satisfies 0<=w<=1
% c is an rgb vector
% B is a uint8 color array with the property that imshow(B) looks like
% imshow(A) with a border. The border color is specified by c and
% the width of the border in pixels is the nearest integer to
% w times the smaller of nA and nB.
```

so that the script

```
A = imread('CornellSnow','jpg');
A_with_Border = AddBorder(A,.02,[1 0 0]);
imshow(A_with_Border)
```

displays the image



Note: The border is to be added *around* the A-image. It is not to be obtained by changing the values of the pixels near the border of the A-image. Submit your implementation of `AddBorder` to CMS.

1.2 Logo

Implement the function

```
  function A = AddCorner(A1,A2)
% A1 is an m1-by-n1-by-3 uint8 color array that corresponds to an image I1
% A2 is an m2-by-n2-by-3 uint8 color array that corresponds to an image I2
% Assume m2<=m1 and n2<=n1
% A is an m1-by-n1-by-3 uint8 color array that corresponds to an image
% that is obtained by placing I2 in the upper left corner of I1.
```

so that the script

```
X = imread('CornellSnow','jpg');
Y = imread('CornellBear','jpg');
Z = AddCorner(X,Y);
imshow(Z)
shg
```

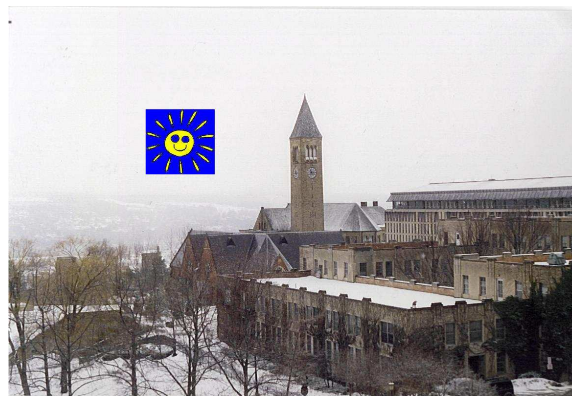
produces the following image:

Submit your implementation of `AddCorner` to CMS.

1.3 Sun

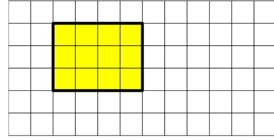
The Undergraduate Admissions Office wants to promote the idea that Ithaca winters aren't so intense after all. To that end they plan to "photoshop" `CornellSnow.jpg` so that it doesn't look so cold. Their idea is to paste the following image into the sky·

(Encoded in `SunBlue.jpg`). Unfortunately, the result isn't too realistic:



Your job is to develop a function `C = AddImage(A,B)` that takes the image encoded by `B` and superimposes it on the image encoded by `A` to produce an encoding of the result in `C`. Your implementation will have two useful features for the problem at hand. It will permit the filtering of the "blue part" of `SunBlue.jpg` and it will enable the user to enter the superposition location in `CornellSnow.jpg` through a mouse click.

The starting point is to develop an intuition for "windowing" and the 2-dimensional "pixel subscripting" that goes along with it. Here is a schematic that illustrates the superpositioning of a 3-by-4 image B on top of a 6-by-12 image A:



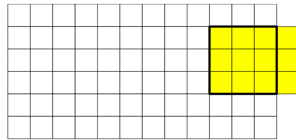
Location: (2,3)

Target: $A(2\!:\!4, 3\!:\!6)$

Source: $B(1\!:\!3, 1\!:\!4)$

Each tile is a pixel. We identify the point of superpositioning by the location of B's upper left corner. Over what A-pixel do we place B's (1,1) pixel? In the example, it is the (2,3) pixel in A. Since B is 3-by-4, we see that $A(2\!:\!4, 3\!:\!6)$ has been "windowed." In particular, $B(1\!:\!3, 1\!:\!4)$ is superimposed over $A(2\!:\!4, 3\!:\!6)$.

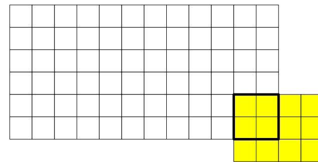
The above example is "lucky" because all of B fits "inside" A. What if the location is such that B spills over the edge? Here are two examples:



Location: (2,10)

Target: A(2:4,10:12)

Source: B(1:3,1:3)



Location: (5,11)

Target: A(5:6,11–:12)

Source: B(1:2,1:2)

The computation of the subscript ranges for the Source and Target involve the row and column dimensions of A and B. Suppose A is m_a-by-n_a, B is m_b-by-n_b, and the location is (i_{loc}, j_{loc}). For a "lucky" superpositioning we have

Source: $B(1\!:\!m_b, 1\!:\!n_b)$

Target: $A(i_{loc}\!:\!i_{loc} + m_b - 1, j_{loc}\!:\!j_{loc} + n_b - 1)$

If the location is near the right edge and/or the bottom edge of A, then the index ranges may have to be adjusted. For example, if $j_{loc} + n_b - 1 > n_a$, then the column ranges for both the Source and Target have to be modified. (See the second example above.)

The next step in the development of `AddImage` concerns the acquisition of the location. The function `ginput` can be used for this but you have to be careful about axis orientation in the figure window when an image is being displayed. Ordinarily, when the figure window houses a plot, the command `[x,y] = ginput(1)`, returns the xy coordinates of the mouseclick. Run the following short script to see the orientation issues that arise when an image is in the figure window:

```
A = imread('CornellSnow','jpg');
imshow(A)
axis on
[x,y] = ginput(1);
hold on
plot(x,y,'*')
```

Note that the mission of the mouseclick is to obtain the location–what we have been calling (i_{loc}, j_{loc}). These are *integers* that identify a pixel. You will have to figure out how to convert from the real values returned by `ginput` to the required integers i_{loc} and j_{loc}.

The last aspect to discuss regarding `AddImage` concerns *filtering*. Recall that we want to "remove the blue" from `SunBlue.jpg`. In other words, we only want to copy "non-blue" pixels from the Source in `SunBlue.jpg` to the Target pixels in `CornellSnow.jpg`. To that end, we need to define the concept of distance between two colors. If C_1 and C_2 are colors with rgb representations $[r_1\ g_1\ b_1]$ and $[r_2\ g_2\ b_2]$, then

$$\texttt{colorDistance}(C_1,C_2) \ = \ |r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2|$$

Thus, if `colorDistance([0 0 1],[r g b])` is "big enough", then the color encoded by `[r g b]` can be regarded as non-blue.

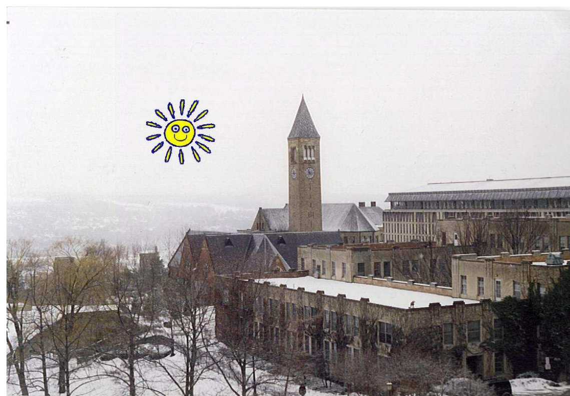
We are now ready to specify `C = AddImage(A,B)`. In particular, implement the function

```
  function C = AddImage(A,B,c,tol)
% A is an mA-by-nA-by-3 uint8 color array that corresponds to an image IA
% B is an mB-by-nB-by-3 uint8 color array that corresponds to an image IB
% C is an mA-by-nA-by-3 uint8 color array that corresponds to a
% superpositioning of IB on IA. The location of the superpositioning is
% solicited via ginput. A Source pixel in B is copied to its Target pixel
% in A only if colorDist(c,v)>=tol where v is the rgb vector associated
% with the pixel.
```

so that the script

```
tol =  ???????
A = imread('CornellSnow','jpg');
B = imread('SunBlue','jpg');
C = AddImage(A,B,[0 0 1],tol);
imshow(C)
shg
```

produces an image of the form



It is your job to determine a good value for `tol`. Experiment! (The location of the Sun is not important.) Submit your implementation of `AddImage` to CMS.

1.4 Monochrome

The fragment

```
A = imread('CornellSnow','jpg');
B = rgb2gray(A);
imshow(B)
```

displays a black-and white version of image in `CornellSnow.jpg`. The array `B` is a 2-dimensional `uint8` array with the property that $B(i,j)$ is the grayness value for pixel (i,j). Suppose `c` is an rgb vector associated with color c, e.g., `c = [.3 .1 .9]`. The fragment

```
cScaled = (double(B(i,j))/255)*c
```

produces an rgb vector `cScaled` that corresponds to a "darkened" version of the color defined by `c`. If we do this for each pixel in `CornellSnow.edu`, then we obtain a monochrome version of the image. Implement the function

```
function C = rgb2mono(A,c)
% A is an mA-by-nA-by-3 uint8 color array that represents an image IA
% c is an rgb vector.
% C is an mA-by-nA-by-3 uint8 color array obtained by scaling pixel
% (i,j) in IA by double(B(i,j))/255 where B = rgb2(A).
```

so that the script

```
A = imread('CornellSnow','jpg');
c = [255 183 213]/255;
B = rgb2mono(A,c);
imshow(B)
shg
```

produces an image like this:

Experiment with different choices for `c`. Remember that colors in a `uint8` array are integers in the range [0,255] while colors in an rgb vector have type double and are in the range [0,1]. Submit your implementation of `rgb2mono` to CMS.

1.5 Putting it All Together

You now have all the tools necessary to make a postcard like this:

A postcard is defined to be a monochromatic version of `CornellSnow.jpg` with the Bear Logo in the upper left corner, a Sun, and three differently colored borders. Write a scipt `ShowPostCards` that generates 5 different postcards and then "plays them back" with cycling.

Start by visiting the website `http://web.njit.edu/ kevin/rgb.txt.html`. Pick 5 colors that you like and assemble their rgb values in a 5-by-3 matrix `MyColors`. You can then let `MyColors(k,:)` be the monochromatic color of your k-th postcard. You can color the borders any way you want.

Use a 5-by-1 cell array `PC` to house the 3D `uint8` color arrays that define the postcards. For example, `PC{3}` should house the `uint8` color array for your 3rd postcard.

For play back, notice that

```
imshow(PC{3})
pause(1)
```

displays the 3rd postcard for one second. Your script `ShowPostCards` should repeat the one-second displaying of postcards 1,2,3,4, and 5 (in that order) three times. Submit your implementation of `ShowPostCards` to CMS.

2 Center of Population

In this problem you compute the center of population for each state using two different methods. One method computes distances using a "flat Earth" model while the other assumes that the Earth is a sphere. You will download the data for the computation from the US Census Bureau website.

2.1 Getting the Data

Follow these steps to get the data into a `.txt` file on your computer:

1. Go to the home page of the US Census Bureau.
2. At the top, click on "Data".
3. Under "Interactive Internet Data Tools", click on "US Gazateer".
4. Click on " 2010 Census U.S. Gazetteer Files".
5. Click on "Counties" and then finally "Download the National Counties Gazetteer File (163KB)."
6. To facilitate grading, rename the downloaded file "`Counties.txt`"

Look at the first few lines of `Counties.txt` and see how it conforms to the data layout description that is given on the US Census Bureau website.

2.2 Accessing the Data in Matlab

Drag `Counties.txt` into the current working directory and execute the command

```
A = importdata('Counties.txt','\t')
```

The file "`Counties.txt`" has columns that are separated by tabs. The `importdata` command extracts the data (respecting the tab-defined columns) and loads it into a 2-field structure:

```
A =
        data: [3221x8 double]
    textdata: {3222x12 cell}
```

Note that `A.textdata` is a cell array and `A.data` is a matrix. There are 3221 counties and information about them is systematically housed in `A.textdata` and `A.data`. For each county, figure out where you can find (a) the 2-letter abbreviation of the state where the county resides, (b) the county's population, (c) the county's latitude, and (d) the county's longitude[1].

2.3 Background Math

As a warm-up exercise, suppose we have a system of N objects in 3-space with the property that object i has weight w_i and is situated at location (x_i, y_i, z_i). The center of gravity for the system is given by $(\bar{x}, \bar{y}, \bar{z})$ where

$$\bar{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} \qquad \bar{y} = \frac{\sum_{i=1}^{N} w_i y_i}{\sum_{i=1}^{N} w_i} \qquad \bar{z} = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i}.$$

Think of these as weighted averages. If w_i is relatively big compared to the other weights, then it has more "impact" on the value of $(\bar{x}, \bar{y}, \bar{z})$. If the weights are roughly the same size, then $(\bar{x}, \bar{y}, \bar{z})$ will be very close to the centroid of the locations.

Center-of-population computations based on Census data are based on the same idea, only the "weights" are populations. Locations are specified by a latitude/longitude pair (θ, ϕ). At location (θ_i, ϕ_i) we have population P_i for $i = 1 : N$.

In the *flat model method* the center of population $(\bar{\theta}_f, \bar{\phi}_f)$ is given by

$$\bar{\theta}_f = \frac{\sum_{i=1}^{N} P_i \theta_i}{\sum_{i=1}^{N} P_i} \qquad \bar{\phi}_f = \frac{\sum_{i=1}^{N} P_i \phi_i}{\sum_{i=1}^{N} P_i}$$

This approach is reasonable if we base the calculations on county populations. Counties are small enough that they we can be reasonably modeled with a latitude/longitude flat map.

In the *globe model method*, we take into consideration the curvature of the Earth. The method involves the following steps:

1. Convert all the latitudes and longitudes to Cartesian coordinates:

   $$x_i = R\cos(\theta_i)\cos(\phi_i) \qquad y_i = R\cos(\theta_i)\sin(\phi_i) \qquad z_i = R\sin(\theta_i)$$

   We assume that $R = 3950$ (miles).

[1] Figuring out how to use an online dataset is an extremely important skill, so appreciate this part of the problem and why you are being left alone to "figure it out!"

2. Compute the Cartesian coordinates of the center of population:

$$\bar{x} \;=\; \frac{\sum_{i=1}^{N} P_i x_i}{\sum_{i=1}^{N} P_i} \qquad \bar{y} \;=\; \frac{\sum_{i=1}^{N} P_i y_i}{\sum_{i=1}^{N} P_i} \qquad \bar{z} \;=\; \frac{\sum_{i=1}^{N} P_i z_i}{\sum_{i=1}^{N} P_i}.$$

3. Obtain the population center $(\bar{\theta}_g, \bar{\phi}_g)$ by projecting $(\bar{x}, \bar{y}, \bar{z})$ onto the sphere:

$$\bar{\theta}_g \;=\; \arcsin(z/R) \qquad \bar{\phi}_g \;=\; -\arccos(x/R)/\cos(\bar{\theta}_g);$$

Note: inverse trig computations are tricky, but these recipes work for the latitudes and longitudes associated with the county data.

2.4 Implementing `CenterOfPop()`

The 3221 counties are distributed over the 52 states. (We count the District of Columbia and Puerto Rico as states in this problem.) Complete the implementation of

```
  function CenterOfPop()
A = importdata('Counties.txt','\t');
         :
```

so that it prints a nicely formatted table with 52 rows and 8 columns:

Column 1: The 2-letter abbreviation of the state. (Use `%s`).
Column 2: The state's population. (Use `%8d`.)
Column 3: The number of counties in the state. (Use `%4d`.)
Column 4: The value if $\bar{\theta}_f$ for the state. (Use `%8.3f`.)
Column 5: The value if $\bar{\phi}_f$ for the state. (Use `%8.3f`.)
Column 6: The value if $\bar{\theta}_g$ for the state. (Use `%8.3f`.
Column 7: The value if $\bar{\phi}_g$ for the state. (Use `%8.3f`.)
Column 8: The great circle distance (in feet) between the two population centers. (Use `%8d`.)

If (x_1, y_1, z_1) and (x_2, y_2, z_2) are situated on the sphere $x^2 + y^2 + z^2 = R^2$, then the great circle distance between them is given by

$$d = 2R \cdot \arcsin\left(\frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}}{2R}\right).$$

Comments. (a) The table should have a nice heading. (b) `CenterOfPop()` should assume that `Counties.txt` is in the current working directory. (c) Remember that the MATLAB trig and inverse trig functions deal with radians while the latitude and longitude data in `Counties.txt` is in degrees. (d) Part of your score will be based on making use of intelligently designed subfunctions. (e) You should vectorize all summation computations. Thus, a weighted sum like

```
s = 0;
for k=1:N
   s = s+w(i)*v(i);
end
```

should be computed as `s = sum(w.*v)`. Submit `CenterOfPop.m` to CMS

Challenge 5: The Big Dissolve

It is possible to make movies in MATLAB. Typing `help movie` and `help getframe` will show you the tools that you need for this Challenge exercise. In this problem you write a function that makes a movie that "dissolves" a given color image into a monochromatic version of the same image in a specified time:

```
  function M = Dissolve(TheImage,TheColor,TheDuration)
% TheImage is a string that names a jpeg image in the current workspace,
%    e.g., 'CornellIthaca.jpg'
% TheColor is a 3-vector that specifies the color of the monochromatic version
%    that is the "destination" of the dissolve, e.g., [1 0 0 ].
% TheDuration is the time in seconds that the dissolve is to take assuming that
%    the movie M is played at 10 frames per second.
```

Make use of the function `rgb2mono` developed from the Postcard problem. Submit `Dissolve` to CMS. Here is a sample test script:

```
% Make a  movie that dissolves CornellIthaca.jpg into a red version...
M = Dissolve('CornellIthaca.jpg',[1 0 0],5);
% Play the movie 3 times at 10 frames a second...
movie(M,3,10)
```

`CornellIthaca.jpg'` can be downloaded from the course website but you can use any jpeg that you want.