CS1112 Fall 2010 Project 4 due Monday, Nov 1, 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.

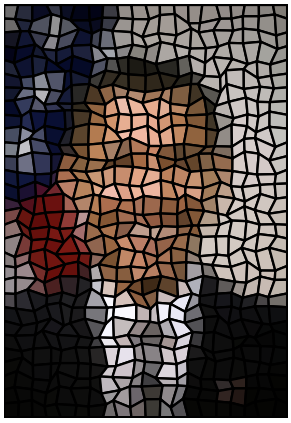
Objectives

Completing this project will solidify your understanding of 2-dimensional and 3-dimensional arrays. You will also work with the jpeg image format and the type `uint8`. Pay attention to the difference between `uint8` and MATLAB's default type `double`.

1 Stained Glass Production Presents...

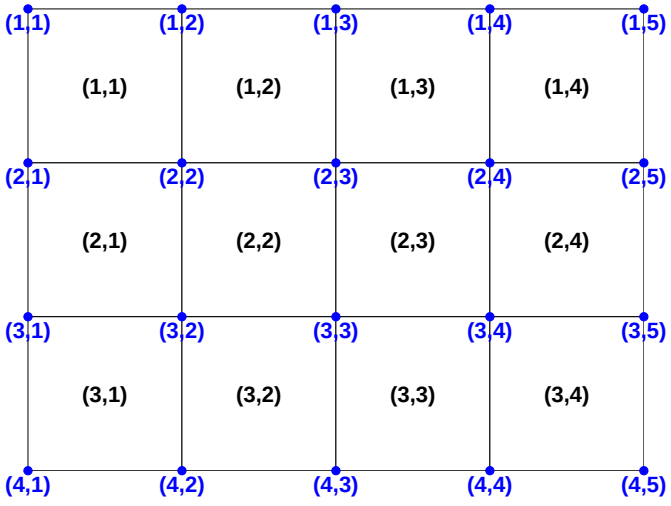
1.1 Background

This project is inspired by a series of posters in the 2008 presidential election. (Do an online search on "Obama posters" to see some examples.) Instead of conforming to "Warholian ascetic," you will write a set of functions to produce a stained glass version of an jpeg image.



1.2 Tiles vs. Grid Points

We need to develop a system for relating the glass tile numbering to the numbering of the grid points. In the regularly spaced grid shown on the right, there are 3 rows × 4 columns of rectangular tiles. The tiles are numbered in black using matrix index notation: (r,c) is the tile in row r, column c *of tiles*. The four corners of each tile are on "grid points." For 3 rows by 4 columns of tiles, there are (3+1) × (4+1) grid points. The grid points and their numbering are drawn in blue on the diagram. *Notice that the tiles and the grid points are numbered separately!* As you write code remember this: in general there are $n-1$ intervals among n points on the number line.



1.3 Computing the color of all tiles

Each glass tile has a color. We estimate the color of each glass tile by computing the "average" color of the block of pixels in the tile area. We do this estimation using rectangular tiles. Download the file `stainedGlass.m` from the Project page. Read the partially implemented function `stainedGlass`, which is the "driver function," or the function that we call to start the process of producing a stained glass figure. Read the specification (function comment) carefully.

The part that you need to complete in `stainedGlass.m` involves dividing the image into `nr` rows × `nc` columns of blocks of pixels and calculating the average color for each block. Each block should be roughly the same size and some rounding likely will be needed to cover all the pixels in the image.

The 3-d array `colr` of rgb values that you need to create has the type `double`, not `uint8`. Therefore you need to map the range [0,255] to the range [0,1]. The "average color" of a block is made up of three values: the average of the red layer, the average of the green layer, and the average of the blue layer. Beware of `uint8` arithmetic!

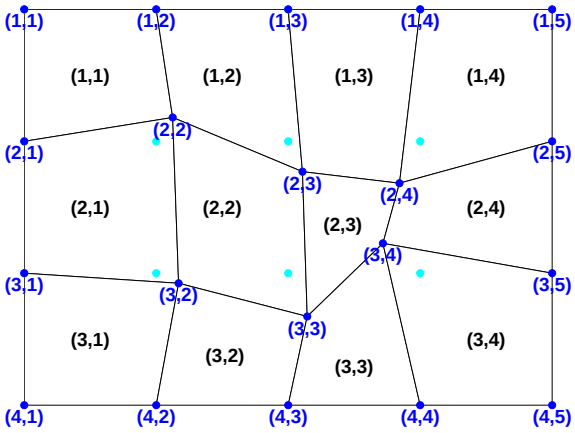
The final two statements in function `stainedGlass` are calls to two functions that you need to implement.

1.4 Coordinates of irregularly shaped tiles

We think that irregularly shaped quadrilaterals are more interesting than rectangular tiles when it comes to a stained glass production! In order to draw each quadrilateral (tile), we need the coordinates of every corner (vertex) of the tile. You will compute all the coordinates of the irregular grid in function `allCoordinates`:

```
function [x, y] = allCoordinates(w,h,nx,ny)
% Generate the coordinates of grid points covering a rectangle of width w,
% height h, and lowerleft corner at (0,0).  nx points cover the width and
% ny points cover the height.
% Start with a regularly-spaced grid.  Then add random noise to the
% coordinates of interior grid points to represent "tiles" that are
% irregularly shaped quadrilaterals.  For each interior grid point, the
% change (in the x and y directions) should be no bigger than 40% of the
% spacing in the regular grid.
% x and y are ny-by-nx matrices such that
%   the x-coordinate of grid point (i,j) is stored in x(i,j) and
%   the y-coordinate of grid point (i,j) is stored in y(i,j).
% Recall that increasing x-values correspond to increasing COLUMN numbers
% while increasing y-values correspond to DECREASING ROW numbers.
```

In `allCoordinates` you are working with *x-y coordinates* in a figure, not pixels in an image. (This function on its own is not about image processing.) On the right is a diagram showing the final grid points (with noise at interior points) in blue and the original interior grid points of the regularly-spaced grid in cyan. The tiles are labeled in black. The specification for the "noise"—the change in the x and y directions for an interior grid points—says only no bigger than 40% of the spacing in the regular grid. So you can play with this amount. Here is a possibility for a random noise in the x-direction that is smaller than 35% of the horizontal spacing in the regular grid, assuming that `x(i,j)` currently stores the x-coordinate of the grid point (i,j):




```
% Calculate max noise allowed
xMaxNoise= .35*horizontalSpacing;  % You need to first calculate horizontalSpacing
% Change x(i,j) by an amount in the range (-xMaxNoise,xMaxNoise)
x(i,j)= x(i,j) + rand*2*xMaxNoise-xMaxNoise;
```

1.5 Producing the stained glass figure

Now it is time to draw the stained glass figure! Again you are dealing with plain MATLAB graphics, not image processing. Implement function `drawAllTiles`:

```
function drawAllTiles(x,y,colr)
% x and y are (nr+1)-by-(nc+1) matrices where nr is the number of rows of
% tiles and nc is the number of columns of tiles:
%   x(i,j) is the x-coordinate of point (i,j) and
%   y(i,j) is the y-coordinate of point (i,j).
% Each point is a vertex of a quadrilateral tile.  Specifically, a tile at
% row r and column c has the vertices at points
%  (r,c), (r+1,c), (r+1,c+1), and (r,c+1)
% colr is a 3-d array of rgb values (size nr-by-nc-by-3):
%   colr(r,c,1) is the red value of tile (r,c),
%   colr(r,c,2) is the green value of tile (r,c), and
%   colr(r,c,3) is the blue value of tile (r,c).
% Draw the nr*nc tiles specified by x, y, and colr.
```

The parameters `x` and `y` store the coordinates of the grid points while `colr` stores the tile colors. So the numbers of rows and columns in `x` are each one greater than the numbers of rows and columns in `colr`.

Use these commands to set up a second window for the stained glass figure, without overwriting the original image shown in figure window 1 (from function `stainedGlass`):

```
figure(2)              % Activate a figure window numbered 2
axis equal off         % Use equal scaling on x- and y-axes; hide axes
axis([0 maxX 0 maxY])  % Fix x-axis to the range [0,maxX], y-axis to the range [0,maxY]
hold on                % Hold subsequent graphics commands on current axes
```

Before you can use the variables `maxX` and `maxY` as shown above you need to first compute their values. To find the maximum value in the `x` matrix, you can write `maxX = max(max(x))`. Why? Since `x` is a matrix, `max(x)` returns a row vector whose kth component is the maximum value in column k of `x`. To get the maximum value of this row vector you call the `max` function again.

Download function `myFill` from the Projects page and use it to draw each tile. Read the specification and code. Function `myFill` differs from the built-in function `fill` in that `myFill` allows the user to specify the thickness of the border that encloses the fill color. For example, the stained glass figure shown on page 1 was drawn with `LineWidth` 2. Feel free to pick the line width that is most pleasing to your eyes.

Use the command `hold off` after all the drawing is done. And you're done with the project as well at this point! (But check your work, of course.) Have fun creating art using your functions!

Submit your files `stainedGlass.m`, `allCoordinates.m`, and `drawAllTiles.m` on CMS.