CS1112 Fall 2016 Project 4 Part B  due Monday 10/24 at 11pm

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. Adhere to the Code of Academic Integrity. For a group, “you” below refers to “your group.” You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student’s code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on your own, seek help from the course staff.

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.

Part A appears in a separate document.

2 Stained Glass Production Presents...

You will give a jpeg photo a “stained glass effect”! We will do this by first dividing the image into non-overlapping blocks of image pixels. As an example, let’s say we have an image with 600 × 500 pixels. Suppose we choose to divide it into 20 × 20 blocks; then each block comprises 30 × 25 pixels. We then calculate a color for each block, which is the average color of the pixels in that block. To simulate our stained glass art, we can treat each block as one “glass tile” of the stained glass. Each tile then has the color of the block. To make the art more interesting looking, we draw each tile as an irregular quadrilateral instead of its original rectangular shape. This process gives us the stained glass effect shown above on the right given the original image on the left (of the fastest man on Earth)!

Blocks (Tiles) vs. Grid Points

We need to develop a system for relating the block 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 blocks. The blocks are numbered in black using matrix index notation: (r,c) is the block in row r, column c of blocks. The four corners of each block (tile) are on “grid points.” For 3 rows by 4 columns of blocks, there are \((3+1) \times (4+1)\) grid points. The grid points and their numbering are drawn in blue on the diagram. Notice that the blocks 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.
2.1 Computing the color of all blocks (tiles)

Each glass tile that we will later draw needs a color. We determine the color of each tile by computing the “average” color of the pixels in each block. Download the files stainedGlass.m and usainBolt.jpg from the Projects page. The given usainBolt.jpg is a small image, which is good for program development. 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.

Blocks

Each block should be about the same size and each pixel in the image must belong to exact one block. For example, suppose we want to divide an image with 600 × 500 pixels into 19 × 20 blocks. Then each block has 500 ÷ 20 = 25 columns of pixels. What about the number of rows of pixels in each block? 600 ÷ 19 gives 31.5789, so one should make some blocks with 31 rows of pixels each while other blocks have 32 rows of pixels each, i.e., the blocks should be very similar in size. One should not make 18 blocks with 31 rows of pixels each, leaving the 19th block with 42 rows of pixels.

The key to dividing the image into blocks of pixels is to calculate the index—row and column index of a pixel, but consider row and column separately—of the first (or last) pixel in each block. Let’s continue to use the example above where an image with 600 × 500 pixels is to be divided into 19 × 20 blocks. The column indices of the first pixel in the blocks are 1, 26, 51, 76, . . . , 476.

Hint: You may find the built-in functions linspace and round useful here in determining the row (and column) index of the first pixel in a block, but there are other ways that do not use those built-in functions. Whichever method you choose, make sure that the blocks are similar in size and every pixel belongs to exactly one block, not more and not fewer.

Color

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!

Summary

So far you have been working with the image data and you have calculated a set of nr × nc colors, one for each of the of nr × nc blocks of pixels. The final two statements in function stainedGlass are calls to two functions that you need to implement. These remaining functions that you will implement deal with general MATLAB graphics, not image processing.

2.2 Coordinates of irregularly shaped tiles

We now create the irregularly shaped quadrilaterals that will each be a stained glass “tile” in our stained glass effect. In order to draw each quadrilateral (tile) using MATLAB’s graphics functions, we need the coordinates of every corner (vertex) of the tile. You will compute all the coordinates of the irregular grid (see diagram below) 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 
play with this amount. Here is a possibility for a ran-
don 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;

2.3 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. `maxX` is the maximum value in the `x` matrix; `maxY` is the maximum value in the `y` matrix.

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. (You can learn more about the built-in function `fill` by typing in the Command Window “doc fill”.)

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

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