CS1112 Spring 2015 Project 4 Part B **due Thursday 3/26 at 11pm**

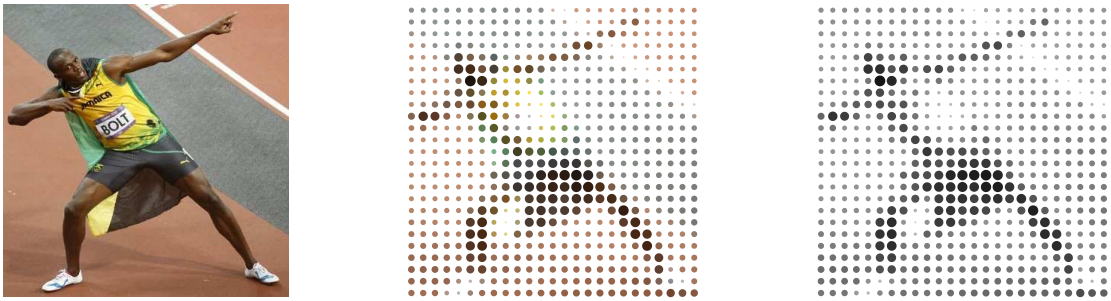
You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies 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 you own, seek help from the course staff.

Objectives

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

Part A (problems 1 and 2) appears in a separate document.

3 Pointillization



Pointillization is a technique of painting using small dots or strokes of color pioneered by French Post-Impressionist painter Georges-Pierre Seurat. We will implement a set of MATLAB functions to "pointillize" digital images. The basic steps are (1) divide the image into non-overlapping square blocks of pixels, (2) calculate for each block the average RGB intensities and average gray intensity, and (3) for each block draw a dot in a size and a color related to the gray and/or color intensity of the block. Our functions allow the user to choose grayscale or full color or anywhere in between. The user can also choose between a still image or an animation—of your own design—of the pointillization.

Download `p4bFiles.zip` from the course website and extract the files into your current working directory in MATLAB. Included in the zip are two function files that you need to complete (`pointillize` and `drawDots`), a completed function that you will use (`DrawDiskNoBorder`), and a jpeg image file that you can use for testing (`usainBolt.jpg`). Fill free to use your own images for testing, but choose *small* ones to use during program development.

3.1 Blocks, color, and grayness

Complete function `pointillize` as specified:

```
function [colr, gray] = pointillize(A,n,t,ani)
%"Pointillize" an image.
% A:  3-d uint8 array of color image data
% n:  Block size, i.e., each block contains n*n pixels (n rows by n cols)
% t:  Grayness threshold below which a block will be presented in color; at
%     or above t a block will be presented in gray.  0<=t<=1.
% ani:  a 0-1 scalar.  1=animate drawing; 0=draw still graphic.
% Crop array A such that its number of rows and number of columns of pixels
% are each a multiple of n.
% Type double arrays colr and gray store RGB and gray intensities, each in
% the range of 0 to 1.
% gray is an nrb-by-ncb array of type double values storing the average
% gray intensity of each block of pixels, where nrb and ncb are the
% number of blocks in the row and column dimensions, respectively.
% colr is an nrb-by-ncb-by-3 array of type double values:
%   colr(:,:,1) store the average red intensity of each block
%   colr(:,:,2) store the average green intensity of each block
%   colr(:,:,3) store the average blue intensity of each block
% The program should halt execution with an appropriate error message if
% the parameter values result in either of these cases:
%   - the number of blocks in either or both dimensions is <10
%   - ani is 1 and the number of blocks in either or both dimensions is >80
```

Halting execution with an error message. You've seen before that MATLAB shows an error message in red and gives an audible "ding" when it encounters a run-time error. We can write our own code to cause program execution to stop under some condition and provide our own, informative, error message. The built-in function to use is `error`; here is an example unrelated to this project that makes use of function `error`:

```
x = input('Enter a positive value: ');
if x<=0
    error('You must enter a positive value!')
end
```

If you execute the above code fragment and enters -3 when prompted, then the program stops, dings, and displays in red in the Command Window the message "You must enter a positive value!"

Crop and block. When necessary, your function should crop from the four edges of the image (data) evenly so that the middle data are kept; if the number of rows (columns) of pixels to crop is odd, then crop more from the bottom (right) than from the top (left). For example, if `A` has the size $159 \times 84 \times 3$ and `n` is 10, then *keep* `A(5:154,3:82,1:3)`, which results in 15 rows by 8 columns of blocks of pixels where each block has 10-by-10 pixels.

Types `uint8` and `double`. The image data (`A`) that we get from a jpeg file have the type `uint8`, but when we use MATLAB graphics later (e.g., with function `plot`) we need the RGB values as a `double` in the range of 0 to 1 where [0 0 0] represents black and [1 1 1] represents white. Notice that the arrays `colr` and `gray` in the function should be in type `double`, which means that you need to scale an integer in the range [0..255] to a real value in the interval [0,1].

Averaging. For each block of pixels, you calculate and store one average red intensity, one average green intensity, and one average blue intensity; the average is done over $\texttt{n}^2$ pixels. To obtain the average gray intensity, take the arithmetic mean of the average red, green, and blue intensities.

Grayness threshold. `t` is passed to and used in another function as shown in the provided code. Given the specification, a value of zero in `t` produces a grayscale graphic. The higher the value of `t` (up to and including 1), the more colors (non-gray) are allowed in the graphic.

3.2 Still graphic of dots

Complete function `drawDots` as specified. The figure window format commands are given in the provided file.

```
function drawDots(colr,gray,t)
% Draw dots (disks) such that the dot radius is inversely proportional to
% the gray intensity of each block (each value in array gray).  A block
% whose gray intensity is >=t is shown as a disk in its gray intensity;
% otherwise the disk is drawn in the RGB color of that block.  The
% background color of the figure is white.
% Type double arrays colr and gray are as described in function
% pointillize.
```

This function uses MATLAB graphics commands to produce a figure—on its own this function is *not* about image processing. Make effective use of function `DrawDiskNoBorder`.

The number of elements in matrix `gray` is the number of dots to draw. The row and column indices of `gray` (and `colr`) serve as the coordinates of the centers of the dots to be drawn: the dot drawn at position (c,-r) has the color `gray(r,c)` or `colr(r,c,:)`. What is the RGB vector of a gray value? Answer: A 3-vector where the components have the same value, e.g., [.1 .1 .1] is a dark gray.

3.3 Animation of dots

Design and implement a function `animateDots` to produce an animated version of `drawDots`. Function `animateDots` has the same parameters as `drawDots`. At the end of the execution of `animateDots`, the figure should be visually the same as that produced by `drawDots` given the same input.

An animation can be as simple as building the figure by sweeping across left to right, but of course we would prefer that you take some time to think about something more interesting. In general, you will need to use function `pause(s)` judiciously where `s` is on the order of 0.01. Keep the animation short!!! In your function comments, in addition to the appropriate documentation that you normally should provide, indicate the parameter values that one should use with the given 389×380 Usain Bolt photo to produce an animation that runs **no more than 10 seconds** on a typical machine, say, an Upson or ACCEL lab machine. Use your creativity **and the techniques and tools that we have covered in the course** in creating the animation. We want you to demonstrate what you have learned so far in the course—don't look up "fancy" built-in functions that we haven't used in the course to do the animation for you.

Have fun "pointillizing" your favorite images! Don't forget to submit your files `pointillize.m`, `drawDots.m`, and `animateDots.m` on CMS.