CS1112 Fall 2014 Project 6 Part C   due Thursday 12/4 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 structs and struct arrays (Part A), object-oriented programming (Part B), and recursion (Part C).

(Parts A and B appear in separate documents. All parts have the same submission deadline.)

3 Generating a Synthetic Sky

[**Before you begin, read** *Insight* **§14.1**; it will help you solve this problem.] Computer generated scenes are used frequently in games and movies. Recursive algorithms see widespread application in the generation of terrain, textured landscape (e.g., flying view of a forest), and sky. For example, the left figure below is a generated Earth sky while the right figure may be the sky of some distant planet, or the scene after the Death Star has exploded . . .

3.1 Algorithm

We will use a modification to the "Midpoint Displacement Method" in two dimensions to generate a "sky scene." The entire scene—a rectangle—is made up of many colored rectangles. We need "random cloud patterns" so that clouds are not the same in one scene and in multiple scenes. We also need neighboring rectangles to have similar but different colors in order to have a natural and smooth looking scene. Therefore the algorithm has both an "averaging" component—similarity with neighbors—and a random component—introduction of variability.

The basic idea is that a rectangle will be filled with the color that is the "average" of its four corner colors. To introduce variability, the average color value is modified ("displaced") by a random amount. This modified average color is called the middle color (as opposed to the corner colors).

The algorithm starts with a rectangle of width w and height h that is the size of the scene. The four corners of the rectangle are each given a color: *cbl*, *cbr*, *ctr*, and *ctl* (Figure 1). The middle color, *cm*, is the modified average color based on the four corner colors (Figure 2). If the rectangle is "big," subdivide it into four equal sized rectangles such that the middle of the original rectangle is a common vertex of the four subrectangles (Figure 3). Each subrectangle needs four corner colors, some of which need to be calculated. For example, the bottom right subrectangle has a bottom-left color *cbm*, which is the average between the two original bottom corner colors, and the top right color *crm*, which is the average between the two original right corner colors (Figure 4). Unlike the calculation of the middle color *cm* of the whole rectangle, there is no random modification to an average color of an edge. For each subrectangle, if it is big subdivide and calculate new colors again. Sudivision continues in the same way until a subrectangle is "small." For a small (sub)rectangle, do not subdivide and instead fill the rectangle with its middle color (*cm*), i.e., draw it in the figure window. The two sky scenes on page 1 were generated using the algorithm as described, with different initial corner colors.

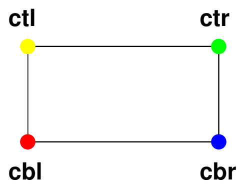

Figure 1

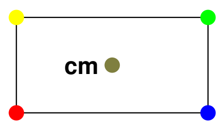

Figure 2

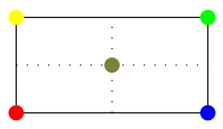

Figure 3

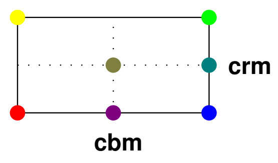

Figure 4

3.2 Function Specification

Implement function `mySky`:

```
function sky(x,y,w,h,cbl,cbr,ctr,ctl,stdev,minsize)
% Draw a sky scene using the recursive midpoint displacement algorithm.
% The scene is a rectangle centered at (x,y) with width w and height h.
% A color is an rgb vector: length 3 and each component is in the range 0 to 1:
%   cbl, cbr:  the colors of the bottom left and right corners, respectively.
%   ctr, ctl:  the colors of the top right and left corners, respectively.
% stdev:  standard deviation of the normal random modification to the middle color.
% minsize:  Fill (draw) the rectangle with the middle color if either w or
%   h is less than minsize; otherwise subdivide the rectangle, using the
%   midpoint as the common vertex, into four subrectangles and proceed recursively.
```

Note that the corner colors are ordered counter-clockwise from the bottom-left. Read the provided script `drawMySky` (on course website) that can be used to call your function `mySky`.

3.2.1 Random Variability

The middle color is the average of the corner colors plus a random value. Use *normally distributed* instead of uniformly distributed random numbers, i.e., use `randn` instead of `rand`. The statement `r=randn` assigns one random number from the standard normal distribution to variable `r`. The standard normal distribution has mean zero and standard deviation one, which says that about 70% of the time the random number would be in the range -1 to +1 (zero plus or minus one standard deviation), with values closer to zero more likely to occur. Given a standard deviation s, scale the value returned by `randn` by s, i.e., `r = s*randn`. In order to create a smooth looking scene, **halve** the standard deviation that is passed to a recursive call of `mySky`. This way more variability is introduced in the beginning when the rectangle is "big"; as the rectangles get smaller—closer neighbors—the variability is decreased.

3.2.2 Color and Drawing

Recall that a color in MATLAB is a vector of length three; each component is a real number in the range of zero to one. You may need to truncate a calculated color value to be in the correct range. Use the `fill` command to draw the rectangle *without* a black outline, e.g.,

```
fill(xpos, ypos, cm, 'line','none')
```

3.2.3 Experimentation and Testing

During initial development of your program, change the value of `minsize` to a *larger* value! For example, set `minsize` to 5 (bigger than either height) to check that your function can draw a single rectangle properly—test the base case. If it works then set `minsize` to a value such that the rectangle subdivides only once—test the recursive case at one level of recursion. If that works than test with an even smaller value of `minsize`—test the recursive case at a deeper level of recursion. At what point does MATLAB warn you about memory usage? (Don't get too carried away with your testing; you may crash your computer!) How much time does sky generation take when `minsize` is .1? How about a `minsize` of .05? What is the effect of a smaller standard deviation? What if you use the *median* instead of the *mean* when calculating the middle color?

You should think about and experiment with the above questions, but you don't have to submit a written answer. Have fun generating sky scenes!

Submit your file `sky.m` in CMS before the deadline. Make sure that your submitted function uses the mean, not median, in calculating the middle color.