
Subject: Help with program speed (Recursive Merging Function)

Posted by [crd319](#) on Tue, 08 Jul 2008 19:48:49 GMT

[View Forum Message](#) <> [Reply to Message](#)

I am working on a texture segmentation algorithm. It works by segmenting an image into smaller "sub-blocks" of varying size based on textural features, comparing neighboring "sub-blocks" and merging those blocks with the lowest "Merger Importance Value". I can get the splitting done relatively fast (4 seconds for a 512x512 image). Im having a problem with my Merging function. Let me explain:

I have my project set up so that it segments the image into an array of pointers, which reference a structure as such

```
*pt[i] = {block, x1,x2,y1,y2}
```

where block is the matrix of pixel values in the sub-block and x1,x2,y1,y2 are locations in reference to the original image.

What I am trying to do is find each neighboring sub-block in the array (for my 512x512 example contains 208 elements). I do this by comparing a cell , pt[i], with every possible match, pt[j] and reading the x,y coordinates. It then calculates the MI value and (will ultimately) put those cells with the lowest MI into a single element (pointing to an array of pointers)

The code works but operates SLOW! It takes ~400 seconds to run through once for my 512x512 image.

NOTE: This is set up for merging single cell but will be used to work with multiple cells at once. ie. 2 cells that have previously been merged are merged with a third.

ALSO: I know I dont have the best programming structure. Im not in a Computing Major and our (my Majors) programming exposure is in Functionality not Efficiency.

```
FUNCTION Merger_importance, pt
t = systime(1)
bins = 8
values = {block1:ptr_new(), block2:ptr_new(), MI:0.}
```

```
MI_arr = [values]
```

```
for i=0, n_elements(pt)-1 do begin
  x1_1 = [-1]
  x1_2 = [-1]
  y1_1 = [-1]
```

```

y1_2 = [-1]

t1 = *pt[i]
blk1 = t1.block
x1_1 = [x1_1,t1.x1]
x1_2 = [x1_2,t1.x2]
y1_1 = [y1_1,t1.y1]
y1_2 = [y1_2,t1.y2]

x1_1 = x1_1[1:*]
x1_2 = x1_2[1:*]
y1_1 = y1_1[1:*]
y1_2 = y1_2[1:*]
p1 = n_elements(blk1)
LBP_C1 = LBP_OVER_C(blk1)
hist1 = HIST_2D(LBP_C1.LBP, LBP_C1.C, max1=255, max2=bins-1)

for j=0, n_elements(pt)-1 do begin
    if (i ne j) then begin
        x2_1 = [-1]
        x2_2 = [-1]
        y2_1 = [-1]
        y2_2 = [-1]
        t2 = *pt[j]
        blk2 = t2.block
        x2_1 = [x2_1,t2.x1]
        x2_2 = [x2_2,t2.x2]
        y2_1 = [y2_1,t2.y1]
        y2_2 = [y2_2,t2.y2]

        x2_1 = x2_1[1:*]
        x2_2 = x2_2[1:*]
        y2_1 = y2_1[1:*]
        y2_2 = y2_2[1:*]

        p2 = n_elements(blk2)
        LBP_C2 = LBP_OVER_C(blk2)
        hist2 = HIST_2D(LBP_C2.LBP, LBP_C2.C, max1=255, max2=bins-1)

        for k = 0, n_elements(x1_1)-1 do begin
            for l = 0, n_elements(y1_1)-1 do begin
                for m = 0, n_elements(x2_1)-1 do begin
                    for n = 0, n_elements(y2_1)-1 do begin

                        if (((x1_2[k]+1 eq x2_1[m]) and ((y1_1[l] le
y2_1[n]) and (y1_2[l] ge y2_2[n]))) or $
                            (((x1_1[k] le x2_1[m]) and (x1_2[k] ge
x2_2[m])) and y1_2[l]+1 eq y2_1[n])) then begin

```

```

        p = min([p1,p2])
        hist1 = hist1+1
        hist2 = hist2+1
        G = abs(g_stat(hist1, hist2, bins))
        struct = {block1: pt[i], block2:pt[j], MI:p*G}
        MI_arr = [MI_arr, struct]
    endif
endfor
endfor
endfor
endfor

endif
endfor
print, i,"Run time: ", systime(1)-t
endfor
Return, MI

END

```

If anyone can help speed this up, I would greatly appreciate it. Or even give me a better idea for my searching routine other than nested for loops.

I know FOR loops in IDL are slow but I couldnt think of another way to organize this.
