Subject: Re: Please help me avoid loops and conditionals
Posted by pford on Wed, 10 Sep 2003 16:33:02 GMT
View Forum Message <> Reply to Message

Well, yours did not work (as you stated that you had not tried it yet), but I did not spend any time on it after I noticed that you forgot to place the origin in the center of the array. Instead I debugged mine. But I have 2 follow up questions. Is it faster to use integers the way I did it then convert to real (floating point) or start off in floating point as you did? My mindset is still that "integer is faster", which may not be the case because of the conversion and the improved floating point hardware, like Altevec on the PPC.

The second is related to the same program. I want to take a string of 4 contiguous bytes and convert them into a long unsigned or signed integer. I remember doing this in Fortran, Pascal and C with some variations on a theme of having variables of type byte and long point to the same memory location. Is there some clever way of doing the same in IDL? My current klugy method is based on the following:

R = ary(p) + (16 * ary(p+1)) + (16^2 * ary(p+2)) + (16^3 * ary(p+3))

Where ary is a byte array. The endian is irrelevant to what I want to do here.

Thanks

Patrick Ford, MD
Baylor College of Medicine
Department of Radiology

Craig Markwardt <craigmnet@cow.physics.wisc.edu> wrote in message news:<onbrttlpy5.fsf@cow.physics.wisc.edu>...
> pford@bcm.tmc.edu (Patrick Ford) writes:
>> function elp2, a, b, box_dim, vval, e_a,e_b, I_ratio
>> x_box = box_dim/2
>> box = intarr(box_dim,box_dim)
>> o_val = fix(vval / I_ratio)
>> v = fix(vval)
>> for i = 0, box_dim-1 do begin
>> for j = 0, box_dim-1 do begin
>> x = float(i - x_box)
>> y = float(j - x_box)

```
>>          if( ((x/(a+e_a))^2 + (y/(b+e_b))^2) LE 1.0) then $
>>    if( ((x/a)^2 + (y/b)^2) LE 1.0) then box(i,j) = o_val $
>>    else box(i,j) = v
>>   endfor; j = 0, box_dim-1 do
>>  endfor; i = 0, box_dim-1 do
>>  return, box
>>  end
>>
>>
>>  So how do I go about converting this into a Boolean matrix operation
>>  that avoids all of this? Would it be faster to create a mask array
>>  such as:
>>
>>   x = float((indgen(box_dim) - box_dim/2) # replicate(1, box_dim))
>>   y = (transpose(x) / b)^2
>
>  It's close to what I would do.
>
>  I would start by creating the X and Y arrays more or less as you have
>  done:
>    x = (fltarr(box_dim)+1) ## findgen(box_dim)
>    y = findgen(box_dim) ## (fltarr(box_dim)+1)
>  and then initializing the array to zeroes:
>    box = intarr(box_dim,box_dim)
>  then as you said, use the WHERE statement to pull out the values of
>  interest.
>    wh = where( (x/(a+e_a))^2 + (y/(b+e_b))^2 LE 1.0, ct)
>
>  ... and fill them in.  You appear to have a two stage process.
>    if ct GT 0 then begin
>       box(wh) = o_val
>       wh1 = where( (x(wh)/a)^2 + (y(wh)/b)^2 LE 1.0, ct1)
>       if ct1 GT 0 then box(wh(wh1)) = v
>    endif
>
>  Just as you, I didn't test this or nuthin'.
>
>  Good luck,
>
>  Craig
```
