Subject: Re: Expensive loops... can they be avoided?
Posted by Allan Whiteford on Tue, 26 Feb 2008 14:15:21 GMT
View Forum Message <> Reply to Message

Rainer,

Not using even a remotely similar algorithm (and one which might be inadequate for your purposes) but how about:

```
xp=transpose(rebin(r,n_elements(r),n_elements(chi))) *     $
          rebin(cos(chi),n_elements(chi),n_elements(r))
yp=transpose(rebin(r,n_elements(r),n_elements(chi))) *
          rebin(sin(chi),n_elements(chi),n_elements(r)) $
triangulate,xp,yp,tr
b=trigrid(xp,yp,transpose(a),tr,xout=x,yout=y)
```

this should be faster than your previous solution but, as I said, might not be suitable. You can also save xp, yp and tr between calls if your "a" data is changing but everything else is staying the same which will also speed things up a bit.

Also, this solution might not work. I've only really checked it for syntax errors not for accuracy or it even being the correct thing to do. I've only ever used triangulate and friends for display purposes so can't claim to be an expert on how accurate "b" will end up - since you were just picking the nearest point before I guess you're not overly concerned with accuracy.

The "missing" keyword to TRIGRID might handle your environment_value case.

Sorry for not answering your question as such but I hope this helps or at least points you in a helpful direction.

Thanks,

Allan

Rainer wrote:
> Hi!
>
> I need to render 2D slices cut from 3D spherical grid data. So far, I
> am using a brute force method which, using two FOR loops, is
> unfortunately rather slow (although working just fine). The problem
> and my approach boil down to this:
>
> -- A[i,j] is the input array, with r[i] and chi[j] being curvilinear
> coordinates (not necessarily uniformly spaced)
> -- B[k,l] is the output array with x[l] and y[l] being uniformly

```
>  spaced Cartesian coordinates (to be plotted with TVSCALE)
>
>  for k=0,nx-1 do begin
>     for l=0,ny-1 do begin
>
>        curr = sqrt(x[k]^2+y[l]^2)
>        curchi = atan(x[k],y[l])
>
>        if "curchi or curr are out of bounds" then begin
>           B[k,l] = environment_value
>           continue
>        endif
>
>        foo = min( r - curr ,nearestr,/absolute)
>        foo = min( chi - curchi ,nearestchi,/absolute)
>
>        B[k,l] = A[nearestr,nearestchi]
>
>     endfor
>  endfor
>
>
>  The calculation of "curr" and "curchi" can easily be done outside the
>  loops of course. But what can I do with the rest?
>
>
>  Thanks,
>  Rainer
```

Subject: Re: Expensive loops... can they be avoided?
Posted by Rainer on Tue, 26 Feb 2008 16:15:19 GMT
View Forum Message <> Reply to Message

Thank you Allan. The TRIGRID solution works, and is amazingly fast,
but unfortunately not quite what I want.

> not be suitable. You can also save xp, yp and tr between calls if your
> "a" data is changing but everything else is staying the same which will
> also speed things up a bit.

Saving the grid indices in my algorithm (nearestr and nearestchi) into
an array and re-using
them already helps a lot but occasionally they need to be re-
calculated.

> I've only ever used triangulate and friends for display purposes so
> can't claim to be an expert on how accurate "b" will end up - since you

> were just picking the nearest point before I guess you're not overly
> concerned with accuracy.

Also my problem is "only" a displaying issue. The nearest neighbor search was intended, though. I'm visualizing data from numerical simulations and the points in "A" lie at the center of grid cells with the value being representative for the whole cell. An interpolation would give a wrong impression. If the grid is coarse, it should be visible.

Thanks again,
Rainer

Subject: Re: Expensive loops... can they be avoided?
Posted by Spon on Tue, 04 Mar 2008 22:02:57 GMT
View Forum Message <> Reply to Message

On Feb 26, 4:15 pm, Rainer <ren...@arcor.de> wrote:
> Thank you Allan. The TRIGRID solution works, and is amazingly fast,
> but unfortunately not quite what I want.
>
> Thanks again,
> Rainer

Rainer,

here's my version, I'd be interested to hear if it works. The keys are to use WHERE in place of the IF statement (predictably), and the combination of the DIMENSION keyword to MIN with the use of MOD to bring the subscripts generated back to meaningful values.

Sorry it's been so long in coming, hopefully it'll make up for it by being that much faster to run! :-)
Let me know how you get on please,

Good luck!
Chris

```
 ; --- Start of code ---

NX = N_ELEMENTS(X)
NY = N_ELEMENTS(Y)
NR = N_ELEMENTS(R)
NC = N_ELEMENTS(Chi)

XX = REBIN(X, NX, NY)
```

```
YY = TRANSPOSE(REBIN(Y, NY, NX))

RVals = SQRT(XX^2 + YY^2)
RIndex = WHERE(RVals GT RMax, RCount, $
  NCOMPLEMENT = RCompCount)

ChiVals = ATAN(XX, YY)
ChiIndex = WHERE(ChiVals GT ChiMax, $
  ChiCount, NCOMPLEMENT = ChiCompCount)

 ; No good data? Get out now!
IF (RCompCount + ChiCompCount) EQ 0 THEN $
  RETURN, REPLICATE(Environment_Value, NX, NY)

 ; Locate bad data:
IF (RCount + ChiCount) NE 0 THEN BEGIN
  IF RCount NE 0 THEN BEGIN
    IF ChiCount NE 0 THEN BEGIN
      BadIndex = [RIndex, ChiIndex]
      BadIndex = BadIndex[UNIQ(BadIndex, $
        SORT(BadIndex))]        ; If bad R and Chi
    ENDIF ELSE BadIndex = RIndex ; If only bad R
  ENDIF ELSE BadIndex = ChiIndex ; If only bad Chi
ENDIF

RVals = REBIN(RVals, NX, NY, NR, /SAMPLE)
RVals = TRANSPOSE(RVals, [2, 0, 1])

ChiVals = REBIN(ChiVals, NX, NY, NC, /SAMPLE)
ChiVals = TRANSPOSE(ChiVals, [2, 0, 1])

RR = REBIN(R, NR, NX, NY, /SAMPLE)
CC = REBIN(Chi, NC, NX, NY, /SAMPLE)

RDiff = RVals - RR
RFoo = MIN(RDiff, LeastR, /ABSOLUTE, $
 DIMENSION = 1)          ; Calculate R subscripts.
LeastR = LeastR MOD NR    ; Bring subscripts back
                          ; to 0:(nr)-1 range.

CDiff = ChiVals - CC
ChiFoo = MIN(CDiff, LeastChi, /ABS, DIM = 1)
LeastChi = LeastChi MOD NC

 ; Define B array:
B = A[LeastR, LeastChi]

 ; Replace any out-of-bound subscripts:
```

```
IF N_ELEMENTS(BadIndex) NE 0 THEN $
 B[BadIndex] = Environment_Value

 ; --- End of code ---
```