Subject: Re: Chunk Array Decimation
Posted by JD Smith on Thu, 03 Oct 2002 20:32:27 GMT
View Forum Message <> Reply to Message

On Thu, 03 Oct 2002 01:58:13 -0700, Craig Markwardt wrote:
> JD Smith <jdsmith@as.arizona.edu> writes:
>
>> On Tue, 01 Oct 2002 14:34:21 -0700, Wayne Landsman wrote:
> [...]
>>>
>>> My solution to the problem combined the REVERSE_INDICiES aproach of
>>> JD, with the "accumlate based on the index" approach. For the
>>> drizzle problem, one is probably only going to sum at most 3-4 pixels
>>> together, so it makes sense to loop over the number of distinct
>>> histogram values (i.e. loop only 3-4 times).
>>>
>>> My solution is below, but I have to admit that I haven't looked at it
>>> for a while.
>>>
>>>
>>> h = histogram(index,reverse = ri,min=0,max=N_elements(vector)-1)
>>>
>>> ;Add locations with at least one pixel
>>> gmax = max(h) ;Highest number of duplicate indicies
>>>
>>> for i=1,gmax do begin
>>> g = where(h GE i, Ng)
>>> if Ng GT 0 then vector[g] = vector[g] + values[ri[ri[g]+i-1]]
>>> endfor
>>>
>>> end
>>
>> That's a very interesting approach, Wayne. People who need to
>> understand the reverse indices vector would do well to study this one.
>> I put it into the same terms as my problem for testing:
>>
>> mx=max(inds)
>> vec5=fltarr(mx+1)
>> h=histogram(inds,REVERSE_INDICES=ri,omin=om) gmax = max(h)
>> ;Highest number of duplicate indicies for j=1,gmax do begin
>> g = where(h GE j, Ng)
>> if Ng GT 0 then vec5[om+g] = vec5[om+g] + data[ri[ri[g]+j-1]]
>> endfor
>>
>> I was interested to see that your method beat mine for normal densities
>> by about a factor of 2! This should provide some cannon fodder for
>> Craig in his loop-anti-defamation campaign: keep loops small, and
>> they're not bad. The only change I added was using OMIN as opposed to

>> fixing MIN=0, but that shouldn't account for much if any improvement.
>>
>> However, one thing still bothered me about the your method: even though
>> the loop through the bin depth is small (e.g. maybe up to 5-10 for
>> DRIZZLE-type cases), you're using WHERE to search a potentially very
>> large histogram array linearly each time. What's the solution? Why,
>> just use another histogram to sort the histogram into bins of repeat
>> count, of course. Now this is a true histogram of a histogram.
> [...]
>
> Here I come late to the game again. This topic actually came up before
> by Liam Gumley in September 2000.
>
> My solution then was the following loop (expressed in today's variable
> names):
>
> n = n_elements(vec)
> hh = histogram(inds, min=0, max=n-1, reverse=rr) wh = where(hh GT 0) &
> mx = max(hh(wh), min=mn) for i = mn, mx do begin
> wh = wh(where(hh(wh) GE i, ct)) ;; Get IND cells with GE i
> entries vec(wh) = vec(wh) + data(rr(rr(wh)+i-1)) ;; Add into the
> total
> endfor
>
> This is essentially the same as Wayne's FDRIZZLE routine, with the
> difference that the WHERE-generated index array is slowly whittled away
> by repeated thinning. Thus, the WHERE() function gets faster and faster
> as the loop proceeds. At the time, I was crowned the victor by Pavel
> :-), but I don't know how I will do against this round of competitors.

Too much fun. I translated your thinned WHERE() method into my terms:

```
mx=max(inds)
vec7=fltarr(mx+1)
h = histogram(inds,OMIN=om,REVERSE_INDICES=ri)
wh = where(h GT 0)
mx = max(h[wh], min=mn)
for j=mn,mx do begin
  wh=wh[where(h[wh] GE j)]  ; Get IND cells with GE i entries
  vec7[om+wh]=vec7[om+wh] + data[ri[ri[wh]+j-1]] ; Add into the total
endfor
```

> However, all of these optimized techniques that Wayne and JD have
> proposed in the end game here, including mine, suffer if the dynamic
> range of the histogram is very large. For example, if the input array
> contains a million 1s, then any of the proposed loops will still take 1

> million iterations. There are even ways around that, which reminds me
> to finish an old routine named CMHISTOGRAM...

With a million 1's, you have only one iteration in your loop, since there's just one bin in the histogram. This example illustrates an error in your formulation: it only works if mn is 1 (which it almost always will be in a large enough vector of random indices)! Why? Because you need the loop to accumulate all of the values from ri[wh]...ri[wh]+n_bin. If you have only one bin of 1000000, you just pick out the value at ri[ri[wh]+1000000]! It's fast, but wrong. FDRIZZLE works correctly because it starts its loop explicitly at 1. Yours works if I modify it to start at 1 also:

```
mx=max(inds)
vec7=fltarr(mx+1)
h = histogram(inds,OMIN=om,REVERSE_INDICES=ri)
wh = where(h GT 0)
mx = max(h[wh],min=mn)
for j=1,mx do begin
  wh=wh[where(h[wh] GE j)]  ; Get IND cells with GE i entries
  vec7[om+wh]=vec7[om+wh] + data[ri[ri[wh]+j-1]] ; Add into the total
endfor
```

In the pathological case of 20,000 1's, I get:

```
WHERE loop:                         0.0014
Literal Accumulate Loop:              0.0246
Reverse Indices Loop:                 0.0014
FDDRIZZLE Loop:                       0.2256
Dual Histogram Loop:                  0.0030
Thinned WHERE Histogram Loop:               0.2623
```

The WHERE loop and reverse indices are essentially equivalent to one call to total with a vector of all indices, and so are quite fast. My method also uses total, but just has to skip all the empty bins. I changed it to do this by starting at min(h1) (rather than just loop through and CONTINUE all those times), and it's fairly fast.

In a more reasonable case of an index density of 5 (indices repeated 5 times on average), I get:

```
WHERE loop:                         0.9506
Literal Accumulate Loop:              0.0245
Reverse Indices Loop:                 0.0213
Loop-Free with Sparse Arrays:               0.0102
FDDRIZZLE Loop:                       0.0064
Dual Histogram Loop:                  0.0040
Thinned WHERE Histogram Loop:               0.0069
```

Strangely, yours always performs slightly worse than Wayne's, despite the thinning. This is a dual processor machine, so your mileage may vary, but in any case it's not faster. Just for fun, here's a run with 1,000,000 random indices with a density of 20:

Literal Accumulate Loop:	1.2437
Reverse Indices Loop:	0.7192
Loop-Free with Sparse Arrays:	1.1367
FDDRIZZLE Loop:	0.7882
Dual Histogram Loop:	0.5489
Thinned WHERE Histogram Loop:	0.8438

If you'd like to try this test code yourself, it's available at:

turtle.as.arizona.edu/idl/

I'd be interested to hear how others find the algorithms stack up.

JD
