Subject: Re: Chunk Array Decimation
Posted by JD Smith on Thu, 03 Oct 2002 00:03:55 GMT
View Forum Message <> Reply to Message

On Tue, 01 Oct 2002 14:34:21 -0700, Wayne Landsman wrote:

>> Of course, anyone familiar at all with histogram() would realize
>> there's a better route when many indices are repeated:
>>
>> mx=max(inds)
>> vec3=fltarr(mx+1)
>> h=histogram(inds,reverse_indices=ri,OMIN=om) for
>> j=0L,n_elements(h)-1 do if ri[j+1] gt ri[j] then $
>> vec3[j+om]=total(data[ri[ri[j]:ri[j+1]-1]])
>>
>> This taps into the ever-so useful reverse indices vector to pick out
>> those elements of data which fall in each "bin" of the index histogram.
>> Notice I'm using OMIN to save time in case the minimum index is
>> greater than 0. This is much faster than the where() method, and can
>> be a factor of 2 or 3 faster than the literal loop approach, if indices
>> are repeated at least a few times on average (a few drops in each
>> histogram bin). If indices are never repeated, or especially if many
>> indices are skipped (a *sparse* set), the literal loop method can be
>> much faster than histogram.
>
> The problem that discussed by JD is actually a very practical one, that
> can be used in "drizzling" algorithms (e.g.
> http://www-int.stsci.edu/~fruchter/dither/drizzle.html) This a
> method of combining or warping images that preserves flux -- every pixel
> in the input image is equally represented in the output image. Instead
> of starting with an input pixel and mapping to an output image (e.g. as
> with POLY_2D) , one starts with an output pixel and determines which
> input pixels get mapped into it. The flux conservation property is
> one very dear to astronomers, and for which there are no existing IDL
> tools.
>
> 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.

```
  mx=max(inds)
  vec6=fltarr(mx+1)
  h1=histogram(inds,reverse_indices=ri1,OMIN=om)
  h2=histogram(h1,reverse_indices=ri2,MIN=1)
  ;; easy case - single values w/o duplication
  if ri2[1] gt ri2[0] then begin
    vec_inds=ri2[ri2[0]:ri2[1]-1]
    vec6[om+vec_inds]=data[ri1[ri1[vec_inds]]]
  endif
  for j=1,n_elements(h2)-1 do begin
    if ri2[j+1] eq ri2[j] then continue ;none with that many duplicates
```

```
    vec_inds=ri2[ri2[j]:ri2[j+1]-1] ;indices into h1
    vinds=om+vec_inds
    vec_inds=rebin(ri1[vec_inds],h2[j],j+1,/SAMPLE)+ $
          rebin(transpose(lindgen(j+1)),h2[j],j+1,/SAMPLE)
    vec6[vinds]=vec6[vinds]+total(data[ri1[vec_inds]],2)
  endfor
```

This is absolutely the fastest I've seen... faster by a factor of ~2 than DRIZZLE. Here are some timings again, for the curious:

20,000 Indices

Indices repeated once, on average:

WHERE loop:	3.8967
Literal Accumulate Loop:	0.0250
Reverse Indices Loop:	0.0725
Loop-Free with Sparse Arrays:	0.0136
FDDRIZZLE Loop:	0.0107
Dual Histogram Loop:	0.0077

Repeated 5 times, on average:

WHERE loop:	0.9433
Literal Accumulate Loop:	0.0241
Reverse Indices Loop:	0.0214
Loop-Free with Sparse Arrays:	0.0102
FDDRIZZLE Loop:	0.0069
Dual Histogram Loop:	0.0041

Repeated 20 times, on average:

WHERE loop:	0.2510
Literal Accumulate Loop:	0.0246
Reverse Indices Loop:	0.0063
Loop-Free with Sparse Arrays:	0.0095
FDDRIZZLE Loop:	0.0075
Dual Histogram Loop:	0.0033

Repeated 50 times, on average:

WHERE loop:	0.1016
Literal Accumulate Loop:	0.0246
Reverse Indices Loop:	0.0032
Loop-Free with Sparse Arrays:	0.0094
FDDRIZZLE Loop:	0.0079
Dual Histogram Loop:	0.0033

Only 1 in 5 indices present (WHERE loop omitted -- too slow):

Literal Accumulate Loop:	0.0275
Reverse Indices Loop:	0.1754
Loop-Free with Sparse Arrays:	0.0453
FDDRIZZLE Loop:	0.0264
Dual Histogram Loop:	0.0196

Only 1 in 20 indices present:

Literal Accumulate Loop:	0.0334
Reverse Indices Loop:	0.4785
Loop-Free with Sparse Arrays:	0.1471
FDDRIZZLE Loop:	0.0623
Dual Histogram Loop:	0.0530

Only 1 in 50 indices present:

Literal Accumulate Loop:	0.0419
Reverse Indices Loop:	1.0674
Loop-Free with Sparse Arrays:	0.3461
FDDRIZZLE Loop:	0.1289
Dual Histogram Loop:	0.1127

Thanks for the pointer.

JD
