Subject: Re: concatenating beyond [[[]]]
Posted by JD Smith on Thu, 24 Oct 2002 08:07:44 GMT
View Forum Message <> Reply to Message

On Wed, 23 Oct 2002 13:08:09 -0700, Marshall Perrin wrote:

> How best to venture into the land where square brackets fail? I want to
> concatenate some 3D data cubes into a 4D dataset. (Basically this is a
> cube of images produced as I vary two parameters in a simulation).
> Square brackets don't work. Is there any more elegant or faster solution
> than just a for loop wrapped around
> bigarr[*,*,i,j] = littlearr[*,*]
> ?
>
> In particular it would be nice to be able to append an entire 3D 'slice'
> to my 4D data cube without having to re-allocate a new larger array and
> copy everything over, but that's probably not easy to do, is it?

Well, given that this is exactly what IDL does when you use the concatenation operator, I think you shouldn't fret. One way to do it without a double loop (which is slow) is just make a new array of the correct size, and drop the data in the appropriate places:

```
d=size(bigarr,/dimensions)
newarr=fltarr([d[0:2],d[3]+1],/NOZERO)
newarr[0,0,0,0]=temporary(bigarr)
newarr[0,0,0,d[3]]=littlearr
bigarr=temporary(newarr)
```

If you find you're doing lots of such concatentations, it will be much faster to pre-allocate a large array and fill it in as you go. This is because IDL is essentially allocating an amount of memory roughly equivalent to N*max(array_size), where N is the number of concatentions (ok, it's really:

$$s+2s+3s+...=s*(n*(n+1)/2))=(n+1)/2*max(array\_size)$$

but close enough).

If you don't know how big the array will be in advance, you can just estimate the size, and keep track of how close to the end you are, allocating more space as needed (one standard method is to double the amount of space added each time). Then, after all is said and done, you trim the array to size. Instead of N*max(array_size), you're allocating more like f*max(array_size), where f is some small multiple (3 seems conservative) depending on the fidelity of your intial estimate and the steepness of the padding size increase .

Example: we end up doing 100 concatention: that's 5050 units of memory
allocated for strict concatenation. Suppose we estimate 20 initially:
we allocate 20,40+20,80+60 for a total of 220. Not too bad.

Good luck,

JD
