

---

Subject: Re: IDL, arrays, and memory  
Posted by [David Fanning](#) on Tue, 04 Feb 2003 16:54:15 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Sean Raffuse (sean@me.wustl.edu) writes:

> I would like to create a jagged array. My array is something like:  
>  
> array = intarr(3600, 1600, 240 or less)  
>  
> Do I save space by creating a jagged array where the 3rd dimension is of  
> variable size? Is that even possible? Should I just stick with  
> 3600x1600x240?

What language are you thinking of doing this in, Sean? :-)

As a general rule, it is better to get all the memory you need at once, then trim. This avoids memory fragmentation problems, etc.

But "jagged arrays". I'd like to see this when you are finished. :-)

Cheers,

David

--  
David W. Fanning, Ph.D.  
Fanning Software Consulting, Inc.  
Phone: 970-221-0438, E-mail: [david@dfanning.com](mailto:david@dfanning.com)  
Coyote's Guide to IDL Programming: <http://www.dfanning.com/>  
Toll-Free IDL Book Orders: 1-888-461-0155

---

---

Subject: Re: IDL, arrays, and memory  
Posted by [JD Smith](#) on Tue, 04 Feb 2003 17:23:50 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 04 Feb 2003 09:54:15 -0700, David Fanning wrote:

> Sean Raffuse (sean@me.wustl.edu) writes:  
>  
>> I would like to create a jagged array. My array is something like:  
>>  
>> array = intarr(3600, 1600, 240 or less)  
>>  
>> Do I save space by creating a jagged array where the 3rd dimension is

>> of variable size? Is that even possible? Should I just stick with  
>> 3600x1600x240?  
>  
> What language are you thinking of doing this in, Sean? :-)  
>  
> As a general rule, it is better to get all the memory you need at once,  
> then trim. This avoids memory fragmentation problems, etc.  
>  
> But "jagged arrays". I'd like to see this when you are finished. :-)  
>

I think Sean probably means "ragged" array, which, in C, is essentially an array of pointers to variable-length arrays, often allocated dynamically at run time. It's a standard memory-saving technique. In fact, when you create a simple static array of strings, ala:

```
char *msg[3]={"Eat", "My", "Grits"};
```

you are implicitly creating a ragged array.

You can easily create such a beast in IDL:

```
array=ptrarr(3600,1600)  
  
for i=0,n_elements(array)-1 do array[i]=ptr_new(vector_less_than_240)
```

However, whereas in C the difference in terms of speed and algorithm design is negligible between using a ragged array and a "wasteful" full 3D array, this is not true in IDL. In particular, you can't use most of IDL's fast array-based operators with an array of pointers; you're stuck accessing each element in a loop, which will be markedly slower for a data structure of this size.

You must balance the memory saved against the speed and flexibility with which you can operate on the data. This is a common theme in IDL, which, in many instances, trades increased memory usage for greater speed of execution. Often you can find other ways to organize the data which reduces the memory footprint while preserving much of the same flexibility had by putting it all in a single array. Or you can use, e.g., NaN values to fill the "wasted" array elements and avoid having to treat them specially.

Good luck,

JD

---

---

Subject: Re: IDL, arrays, and memory  
Posted by [James Kuyper](#) on Tue, 04 Feb 2003 17:29:27 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

David Fanning wrote:

>  
> Sean Raffuse (sean@me.wustl.edu) writes:  
>  
>> I would like to create a jagged array. My array is something like:  
>>  
>> array = intarr(3600, 1600, 240 or less)  
>>  
>> Do I save space by creating a jagged array where the 3rd dimension is of  
>> variable size? Is that even possible? Should I just stick with  
>> 3600x1600x240?  
>  
> What language are you thinking of doing this in, Sean? :-)  
>  
> As a general rule, it is better to get all the memory you  
> need at once, then trim. This avoids memory fragmentation  
> problems, etc.  
>  
> But "jagged arrays". I'd like to see this when you are  
> finished. :-)

In C, you can implement jagged arrays by defining a two-dimensional array of pointers to allocated arrays, where each of the allocated arrays can have a different length. You can write custom functions that know that the length can vary, and they can be very efficient.

In IDL, while you do have pointers, there's such a big speed difference between the native array processing facilities and user-written loops that this is probably not worth the trouble. You're probably better off sticking with 3600x1600x240.

---

Subject: Re: IDL, arrays, and memory  
Posted by [Sean Raffuse](#) on Tue, 04 Feb 2003 17:31:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

>  
> However, whereas in C the difference in terms of speed and algorithm  
> design is negligible between using a ragged array and a "wasteful"  
> full 3D array, this is not true in IDL. In particular, you can't use  
> most of IDL's fast array-based operators with an array of pointers;  
> you're stuck accessing each element in a loop, which will be markedly  
> slower for a data structure of this size.  
>

> You must balance the memory saved against the speed and flexibility  
> with which you can operate on the data. This is a common theme in  
> IDL, which, in many instances, trades increased memory usage for  
> greater speed of execution. Often you can find other ways to organize  
> the data which reduces the memory footprint while preserving much of  
> the same flexibility had by putting it all in a single array. Or you  
> can use, e.g., NaN values to fill the "wasted" array elements and  
> avoid having to treat them specially.  
>  
> Good luck,  
>  
> JD

Exactly what I needed to hear. Speed of execution is indeed a major issue here. I'll stick with the big array.

---