Subject: Re: Yet another object graphics question
Posted by Michael Wallace on Sat, 26 Feb 2005 00:01:38 GMT
View Forum Message <> Reply to Message

>> * FORTRAN doesn't count :-)
>
>
> Wha...?!?!? The cheek! <insert arm flailing and much huffing and puffing>

Okay, I have to admit that I purposely inserted the FORTRAN comment just to see what kind of reaction I could get out of you folks. ;-) Just some good natured kidding -- that's all. :-)

While I do agree with you that FORTRAN has come a long way, many of the scientists who I work with still use F77. Scientists on the cutting edge use F90. The newer crop of scientists don't even touch FORTRAN, they go straight to IDL.

> I would suggest that Java/C++ are not good languages for learning how to
> design software. That should be language independent. (That's my dig. :o)

I believe Java, C and sometimes C++ are the best languages to use when learning how to design software. However, software design should be language neutral. You need to make a distinction between the process of designing software and the implementation of the software. If you're learning software design, you need to learn about stacks, queues, trees, graphs, pointers, objects, memory access and everything else that falls into the conceptual realm. It's possible to talk about and understand these ideas at length without ever having written a line of code. This is great theory, but to be useful and to really understand it, you have to put it in practice. I don't know about everyone else, but my understanding of a concept only really crystallizes once I see it in action and have it reinforce all the theory.

For putting concepts into theory, you have to pick a language to do it.
There is no universal language that's equally good at everything. Therefore, you have to pick a couple languages to use as the "teaching languages" and only divert from those in special cases. When trying to learn concepts, the last thing you want to do is introduce a new language with new syntax and new style. You should be focused on putting the theory into practice rather than learning new syntax. There will be time to learn syntax later -- now is for learning about compilers or whatnot. That said, you'd want to use languages that'd be generally good for most problems and understanding most concepts.

For procedural languages C fits the bill. It's general purpose and there's a lot of good books and information about the language. Also, C is very good for understanding pointers, memory access, operating systems and networking simply because it doesn't hide all the details that other languages do. I'd be willing to bet that the person who has written networking code in C has a much better understanding of the core networking concepts than someone who's written equivalent code in another language such as Java.

For object-oriented programming, Java is currently the best one out there. There's many good resources and since it does hide some of the details of pointers and the like, it allows you to focus in on a specific algorithm or data structure without being bogged down by handling all the memory access yourself. C++ was once used as the standard language for teaching OO, but Java is clearly a better alternative, not only because you don't have to worry about getting your pointer arithmetic wrong, but more importantly C++ lets you break OO if you want to.

With my IDL programming, I know conceptually what I want to do, but now I need to be concerned with the idiosyncrasies of the language itself. IDL objects are not the same thing as Java objects. IDL's objects are more like glorified structures that anything else. There are common ways that I've learned to attack problems in Java that make for a mostly elegant approach while also being pretty quick and make good use of memory. When I first jumped into IDL, I tried using the same approach with IDL and found my IDL chewing up every bit of available memory and grinding to a halt. I've come to learn that there are things which IDL does better and things which IDL does worse. Even though you have a great theory, sometimes that theory doesn't work out very well because of constraints of the language itself. You need to adapt the concepts somewhat in order for you to get the elegant, efficient program on the other end. That's what I'm trying to learn how to do right now and that's what I meant by saying that I needed to write my book on how to write IDL if you've programmed in a real language before.

Anyway, happy trails with whatever language you use, even if it is FORTRAN. ;-)

-Mike