Subject: Re: When should objects be used?
Posted by Struan Gray on Mon, 28 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

This thread seems to have split into two parts: IDL's own built-in object routines and user-written objects. I think you either love or hate the provided object graphics. I'm gaining more and more respect for them, but I think that in general the provided routines are either too high-level or too low-level, and reading the OpenGL specs I feel that RSI has unnecessarily hamstrung their implementation. I like the way object graphics takes care of any colour table juggling for me, and the way that data picking is made substantially easier.

Some problems are intrinsically object-oriented. People have already mentioned minor variations on a theme, such as multiple file-types for the same sorts of data, but there are also data which are naturally hierarchical. I have some crude routines for making and plotting crystal structures. Adding atoms to a unit cell and then specifying a lattice and a bounding polygon is an obvious OOP way to build up crystals. Because the objects keep track of their own properties it is then easy to alter the size or colour of any the individual atoms and the whole crystal instantly reflects the changes. It is also easy to interact with the crystal: to add or remove bonds and coordination polyhedra; to muck about creating point defects and lattice distortions; and to add surface structures, adsorbates or the interface to anther crystal. Toss in the fact that the whole thing can be rendered in 3D with lights, perspective and depth-cueing and you have a pretty persuasive arguement for OOP.

In IDL there are some language-based reasons for using objects. They are persistent, globally available, and easier to clean up than rivals like pointers. You can give novices templates which you know will work reliably with the rest of your application, because you've hidden the dull but necessary behaviour in a superclass that they can't edit. Best of all, you don't *have* to use objects, so you can mix and match behaviour with normal variables, widgets and the command line: you lose none of the prototyping power of IDL.

The only downside is that objects give you a whole new way of writing spaghetti code, since method definitions can be hidden in or distributed among many-times abstracted superclasses. Also, I often get an uneasy feeling that I'm the Sorcerer's Apprentice and the broomsticks are about to make their autonomy felt, but being able to interact with and interrogate objects from the command line helps calm me down.

Struan