DESIGN NOTES FOR A NEW CAD SYSTEM

William Bricken
December 1988

Let's not get efficiency (the cart) in front of modeling (the horse). That is, I vote for classes for all objects except perhaps the very lowest (integers, reals, points). We can always remove classes later, in favor of machine efficiencies.

Some reasons for having vectors, tuples, matrices, etc. as classes:

They are all just structures, and structures are classes

We might want to do abstract algebra on them (ie computations with variables, such as n * (a, b, c) = (na, nb, nc)). Symbolic operations are not faster when implemented as data types.

Data types and classes are different, not interchangable. (I'll need to look up exactly why.)

We must be thinking classes; the main goal is CONCEPTUAL INTEGRITY. The risk of mixing implementation with model is confusion, both in our minds and in our code.

If we really care about efficiency, we'd use intrinsic data-structures or quaternions instead of matrices. What we do want is an *abstraction barrier* between the concept (eg rotation) and the implementation (eg matrices, quaternions, turtles, ...).

Say we have some conceptual entity such as a line. The line object will need to include all forms of its definition (such as point-pair, slope-and-intercept, point-and-slope, etc). If we used only one form of the definition, the object line would get confused when it was referred to by a different form. We then will choose to embody the object line within a mathematical structure (such as geometric-measure, topological-continuity, or even as a set), depending on the operations we will do with the object. Finally, the mathematical structure will find an implementation form, so that sets (for eg) might be Lists, or Collections, or Arrays, etc.

The important idea here is that the conceptual objects are separate from the mathematical models that we formalize them with, and the mathematical objects are separate from implementation objects we use to code them with.

So we need objects such as Strings, Collections, and Structures, but only to the extent that they supply a mathematical context for the geometric concepts, such as Line, 3DCircle, etc. We need programming objects, such as

Arrays, Circular Queues, etc, but only to the extent that we choose to use them as implementation substrata for the mathematical context. And when we step from math to implementation, we can also step from objects to data types.

While we're making object lists, let's remember that an object is both a data structure and a set of operations on that structure. Only by specifying the operations will we be able to identify the appropriate mathematical theory (and thus the appropriate support objects). Ron's list does incorporate mathematics, but lots of changes become apparent when you ask "where's the theory?"

Some quick egs:

> Collections are Sets
> Typed collections (by array, byte, etc) are confused.
> Ordered Collections are not subclasses of Collections
> Sorted Collections are not subclasses of Collections
> Not all Graphics Primitives are Structures.

In general, implementation is mixed with mathematics is mixed with concept.