PROGRAMMING CYBERSPACE

William Bricken
April 1989

Email correspondence from various sources.

Fantasy (william)

My fantasy is an 80860 for each eye, with a parallel auxiliary of a batch (say 16 or so) of Transputers to be assigned to maintenance of specific (complex) objects. Rather than doing collision detection from a single perspective (say the user), each object does parallel "zone" detection, monitoring each moving object within its hierarchical zones of control. You might recognize this as the way the US Early warning radar system is set up. If each object has control of the local space in its vicinity, it could express its preferences by altering the metric of the underlying space in its control. The "space processor" gets a parameter from an dominant object (the owner of the space) whenever a moving foreign object enters its "air space". The parameter might be 1, or unity, which would not change space, or it might be an exponential repulsive force, which would make the approaching object expend more energy the closer it came, or it might be a tractor-beam...

Who owns a chuck of flexible-metric space? Perhaps the biggest object, or perhaps objects might collect power points (a greater sphere of influence). What about just adding the functional parameters each object has in effect on a shared space?

I like to think of the space of Cyberspace as cellular, with a influence vector as an attribute in each cell. Spatial control parameters are one way of changing the influence vector. A very simple way is to subtract "g" , that is have an influence vector pointing down and equal to the force of gravity. So a local gravitational field is implemented by subtracting the influence vector from the acceleration of the mass occupying each point. Statistical averaging and sampling (center-of-gravity, field differential, etc.) makes the computation tractable.

Well, at least, that's the fantasy for now.

Programming (William)

In contrast to standard object-oriented systems, in which the global object is some abstract "thing", the global in Cyberspace is "space" itself. Objects can be composed hierarchically, as would be expected. So sub-objects are spatially dependent on their containing object. The key idea is that objects *within a space* are sub-objects of the containing space. We can then twiddle the characteristics of space (say gravity) to change (by inheritance)

the behavior of all objects within a space.

This rather minor change, from object-centered to space-centered, permits many wonderful algorithms. Movement of any kind, for instance, can be built into the fabric (underlying metric) of space, rather than being added to the characteristics of objects. Space-based reasoning permits overloading of behavior and existence. Changes in coordinate systems (Cartesian to Spherical, for eg) are no longer changes in object representation, instead they are changes in the metric of space.

But most fundamentally, empty space is not Void, rather it contains the attention of the user. Space-based models incorporate the user/programmer/mathematician/cybernaut at the very beginning. In conventional models of interface, the user is always in an external perspective (outside). Space-based models unify object and observer, permitting principled techniques for changing perspective (outside/inside the space).

Choice of external/internal perspective underlies most programming dualities, including object/process, global/local, sequential/parallel, batch/interactive, deterministic/autonomous, and procedural/declarative. The distinction between thing and space makes parallelism easy.

Cyberspace programmer's perspective

We are not assuming, at all, that cyberspace will be "rather static". On the contrary, we're designing it to be fundamentally dynamic, including multiple participants and parallel processes. A fundamental principle is that each control room (physical space from which actions are mapped into cyberspace) will be driven by a computing engine that includes a complete model of the virtual world. In other words, there is only one simulated world, no matter how many nodes in the network. Each node might include hardware to support the parallel execution of multiple bodies (dynamic objects), but that's not what was alluded to; rather, we were pointing out that only state changes need flow around the network. At the network level the problem isn't throughput: it's synchronization.

For many spaces, there may well be a need for a "world master", but we aren't looking at it as the central authority that somehow provides the world to the nodes. Rather it will simply be another object in the world with capabilities suited to the purposes of the world designers. It might, for example, determine which participants (people or programs) can communicate with each other at particular times. The basic idea is to distribute autonomy so that interesting - and unexpected - behaviors can emerge from interactions among participants. Think of it as setting up a context for action rather than a prescription. Interesting worlds aren't likely to be pre-programmed.

Cyberspace class hierarchies (programmer)

Just being curious, I decided to look at the class hierarchies in the cyberspace code. Here's how they look today, subclassing denoted with indentation:

```
Where the action is, container of bodies and sensors
      cyberspace

A graphical entity in cyberspace
      body
            camera

How cyberspace looks (sufficiently different from a body)
      e3view

Lights (could be body subclass)
      light

What the patron of cyberspace wears
      Sensor
            Keyboard
            Sensor2d
                  Joy
                  Mouse
            Sensor6d
                  DataGlove
                  Polhemus
            Dim6

How sensors communicate with the host
      communication
            serialport

Assorted 2d drawing objects
      shape
            e2rect
            e2polygon
            e2polyline
      s2rect

3-d matrox-supported primitives
      e3solid
            block
            sphere
            cone
            cylinder
            torus
```

```
"Spaces" related to management of 2 and 3-d drawing capabilities
      s2space
      e2space
      e3space

Linear Algebra
      vector
            coord
            convector
            euler
            spinor
      matrix

Some board-level software entities
      colorLut
      segment

Classes corresponding to particular hardware components
      matroxBoards
      Textscreen


For numbermongers, the Cyberspace system has
      .hpp class definitions,       1789 lines in 30 modules
      .cpp class implementations,   5565 lines in 29 modules
      .h global constants,           739 lines in 6 modules
```

And more (another programmer)

I'd also add that there's quite a bit of detail within each class, with all sorts of implied conceptual threads. The cyberspace class, for example, is defined recursively. In general, cyberspace will not be just one place, but a space of spaces.

Programming Inside Cyberspace (CAD developer)

But what about programming? Where does that fit into Cyberspace? Is the software vendor back in the position of anticipating everything the clever user might want, and/or limiting him to operations that can be performed by moving things around in 3-space?

For example, suppose we provide the user with a particular magic wand with which he can touch an object and replicate it into an orthogonal array. Then can you imagine the number of other magic wands the users will want to have available and that we won't want to be in the business of creating for him?

Programmers' reply

We have been talking about the spirit of cyberspace, which emphasizes the manipulation of models as if they were real objects. There's nothing in the approach that precludes programming and customization. On the contrary, we're working very carefully to develop a flexible and open architecture. What happens in a space, for example, is entirely up to the dynamic bodies in the space (which are now implemented as C++ objects). The particular bodies won't come from us, by and large, but rather from our customers, who will use our tools to build the bodies. Programming, as always, will be the essential tool. The style of programming, however, will likely evolve in directions fostered by the new paradigm. Things that we haven't normally considered in the past, like parallelism, timing, forces, emergent behavior, and so on, will significantly influence both how we program and what tools we supply. Personally, I think an object-oriented Lisp, integrated with C++, would be ideal for cyberspace development. What we need is a programming capability that helps us DIRECTLY PROGRAM dynamic bodies. I can't imagine anything more at odds with the spirit of cyberspace than traditional compilation (which compels programmers to wait, lose their train of thought, and mentally disengage from the space they're working in). Presently we're working on nuts and bolts (basically an operating system), in C++, but when that's in place we can start pushing the evolution of programming tools. If you're worried that the role of programming may be diminished, don't: it's going to be more important and exciting than ever.

Programming in Cyberspace (William)

Cyberspace is an interface technology and a philosophy of computing. The philosophy is that *strings of symbols* rob us of easy ways to say lots of things. I'd rather point with my finger than type in a symbolic pointer.

It's well established that specialized languages are needed to say specialized things. So the question is "What's the best way to issue instructions in CAD?" The general observation is that strings of symbols are notoriously poor for conveying spatial concepts.

VPL and the Lab have been working on fully expressive visual programming languages. Think of stacking up toy blocks instead of symbolic words to program. It's relatively easy to express abstractions such as GOTO and IF-THEN-ELSE as 2D diagrams and 3D sculptures. Think of parallel processors, in which the only way to capture the asynchronous activity is with a diagram. (Stacks of words, linear dumps, are degenerate projections of parallel computation.)

Sure, someone will need to compress these visual concepts into linear code. That someone is us, the tool is the Cyberspace Construction Kit. We provide

the infrastructure, applications programmers provide the worlds. Building worlds is effort intensive, build tools to build worlds is profit intensive. CAD forms the basis of a world building toolkit. What we do best as a company is to provide world building tools. The Cyberspace Toolkit adds formal models of real world constructions.

So, multiply an abstract template by an object to get an orthogonal array of those objects. We provide the capability for form abstraction, and the ability to point to the object. The user draws the abstract shape (say a 3x4 orthogonal array) and pushes the multiply button. The user can link the instruction to multiply by an abstract form (say, OrthogonalArray) to a wand if desired.

No one will do text processing in Cyberspace, cause text is 1D and is better handled by string processing environments. No one will do form processing in words, cause spatial forms are 2D and 3D, and better handled in CAD and in Cyberspace.

So you can do abstract programming in Cyberspace by placing an abstract semantics on 3D objects rather than 1D words. You can do macro programming in Cyberspace by hooking form abstraction tools onto interface devices. And (contrary to current options), you can choose the environment that best suits the ideas you wish to work with.

And to amplify Peter Schwartz' point: the key to software success is not sales, but *resilience*. When it changes (and it will change), we must be ready to change with it. Fortunately, as a leader, we can define the direction of change actively.

Concerns (CAD developer)

What more does pointing with your hand in the virtual world buy you than pointing with a mouse on a 2D screen projection of a 3D world?

I watched someone go through conniptions with the data glove to grab and move a cube. Isn't it easier to use the small, refined hand movements to move a mouse than it is to wave your arm around?

And isn't a 2D view going to be necessary in your virtual world because of the sheer ambiguity of the 3D world? In which case, if all drafting requires a move to a plan or 2D elevation view, why even bother with cyberspace in the first place?

Some replies (programmers)

> What more does pointing with your hand in the virtual world buy you
> than pointing with a mouse on a 2D screen projection of a 3D world?

What it buys you is directness. You want to specify some 3-d point, you simply point to it. That's all that's needed.

> And isn't a 2D view going to be necessary in your virtual world because of
> the sheer ambiguity of the 3D world?

Huh? Don't you have this backwards? The 2-d projection is ambiguous, in the sense that you've lost information (depth information) in projecting into 2-d. The same 3-d object projects into 2-d in may ways (cf. Nekker cube illusion, etc.)

> I watched someone go through conniptions with the data glove

Permit me an analogy:

I watched you go through conniptions with the mouse to grab and move a cube. Isn't it easier to use the small, refined finger movements to press cursor keys (a 1-d positioning device) than it is to slide your arm around?

For whatever reason, the world I perceive seems 3-d. I'd like the world I interact with in the computer to have the same perceptual and interactive characteristics. If your arm tires moving in 3-space, use a mouse. If your arm tires using a mouse, use cursor keys. But I claim that to do the same thing, you're gonna have to slide more than you wave, and press more than you slide.

> In which case, if all drafting requires a move to a plan or 2D elevation
> view, why even bother with cyberspace in the first place?

Drafting? Who's talking about drafting? I want (and am working to have) direct manipulation of realistic 3-d things in 3-d.

I concur that cyberspace may well be irrelevant to drafting. But when cyberspace matures, drafting will be irrelevant to design, just as clay tablets and pointed sticks are currently irrelevant to writing.

Reply (William)

We assume that we are working with hardware and interface tools that are in their infancy. Yes, the glove is clumsy now, but soon... It's incredible what a large market can do for hardware.

The win in going from mouse to finger is naturalism. How much training does it take to teach a kid to point at something?

3D worlds are ambiguous in CAD because we don't do 3D modeling. There is no ambiguity with a formal model. And clearly, a projection onto a space of lower dimensionality (3D -> 2D) must always lose information. That's what going to a lower dimension means. We are working to increase information.

All your questions seem based in the present rather than the future.