Paul:

The other night I had a rather interesting and illuminating talk with Bob d'Ancona about animation and came up with an interesting fact: it is just about as interesting to watch a picture being drawn (if it is done interestingly) as to watch the result. By concentrating on revealing the process of creating an image rather than on final images we may be able to get our hands on a different flavor of animation.

One way of looking at this style of animation is to take Saul Steinberg's rubber stamp kit and complete a drawing. For example, fill in a movie theatre and then empty it out. Since we can make things disappear via color matrix hackery we can also empty it out. Perhaps a street scene which fills and empties in a process that includes a certain sense of virtual motion of vehicles, people ...

The ideas are probably incomplete but what we can say we are doing is developing an image oriented color organ. The interface could be like that of an organ (only spatial) and in theory we could save and tune good scripts in a player piano fashion.

One implementation involves using the rubber stamp metaphor which allows the construction of a set of stamps in something like paint. Operation one is to copy a stamp much like xerox brush mode. Operation two is to copy with simple scaling or distortion. Operation three is to copy with a new color set. With a pixel window it may be possible to copy a bit faster than normally. (I have some ideas about this based on WCD stuff.)

Another piece of the implementation involves color matrixing and can assume each stamp of a given color has its own matrix, each element of which can be hacked separately or by some rule. In addition the matrices of several different stamps can be catenated which allows more complex rules.

These rules could contain:
copying a matrix element to elsewhere which is a subset of swirling (or rather its component element)
changing a color with time, more or less scripting it - we probably want this done to colors not slots
binding a color or speed change to places on the screen or performance device so that it is possible to play a complex process

For input the TSD would be a nice feature although it would have to be put on the 85 or we will never get to use it except on ceremonial occaisions. The thing I am most worried about is that it is very easy to get very complex and introduce more variables than can be easily tuned so the user interface will be very important. As far as system resources go we should try an be sparing. We might be able to assume some fast overlay space but I personally would rather eliminate huge piles of features in exchange for immediacy.

Let me know what you think.

Seth

EOM - A Slightly Different Animation System

There are a number of ways of approaching the problem of computer generated animation. One is to limit the domain of actions in order to keep the user interface simple. For example, a flight simulator treats animation as a change in the user's point of view. Systems which allow more direct control of the image fall into two main classes:

1) The simulation approach which treats the construction of the image as the simulation of a set of tasks. Thus each piece of the picture is orchestrated via its script.
2) The semanticly based approach which treats the image as the result of a computation which can be expressed as either a hairy equation or as a node network.

Many traditional systems of animation use the former approach since in many ways it approximates the approach used by professional animators. The latter approach has been used in a real-time system by the Chicago Circle Graphics Habitat. We chose to try out the latter approach since it seemed to offer an interesting way of handling complex phenomena. It also vaguely resembled the techniques of analog computing since each node could be considered an object in the user's supply tray.

The semantics of the animation system resemble those of APL in some ways and in some ways are unique. The basic data types are:

1) numbers (floating point)
2) positions (cartesian coordinates)
3) colors (intensity, hue, saturation)
4) shapes (basically a list of points with a color and a "flood point" so it could be filled in)

Operators included such things as rotators which took a shape, a point, and a number and rotated that shape around that point. Other operators performed such operations as addition, subtraction, and the like.

In addition to operators there are source and sink nodes. Source nodes include the clock which contains the current time (actually the virtual time in the sequence), constant values of numbers, points, colors, and more recently shapes. Sink nodes include the screen, the background, and the console for debugging.

Operators are defined to build a point or color out of a set of numbers. Another operator groups a set of points, a flood point and a color to produce a shape. Additional operators exist to extract information about a shape or color.

Normal values can be considered as scalars since scalars can be grouped together to form groups. A group is basically an ordered list of scalars which are usually but need not be all of the same type. This corresponds loosely to the APL idea of an array. When a group is an input into a node that node is invoked once for each value in that group and the values produced by the group are linked together to form a resulting group.

As in APL, if more than one input is a group then all groups going into the node must be of the same length.

So, to have four squares appear at different places on the screen all one needs is an input value of the square and a group of the four points on the screen. The group and the shape are passed to a point-it node and the result is a group of four repositioned squares. (see diagram)

The user interface to the animation system was started by Jim Davis and has been continued by a number of people including Larry DeMar(?) and Dan Franklin. The node network is implemented on the screen of an IMLAC (PDS-1) terminal which by means of a light pen and a set of menues allows the user to create, delete, link, and unlink the nodes in the network. Since there are more nodes known by the system than can fit in the menu area on the screen there is a menu to select which menu to display in the other part of the screen. This requires that the user realize into which class of nodes a given node falls. (For example, add, subtract, and the like are arithmetic nodes while point-it, shape-it, and rotate are shape manipulators.)

The interface provides a set of buttons on the IMLAC which determine how the light pen should act when it touches something. Normally, the light pen can be used to slide things around on the screen. One button turns it into a deleter which can be used to remove nodes and links. Another button allows the pen to select a from node and the other a to node so that links can be created. A final button allows the values of constant nodes to be interogated and modified. Thus, numerical values can be typed in, colors can be chosen in a color editor (or typed in), points can be selected with the light pen and tracking cross, while a shape editor can be used for manipulating constant shapes.