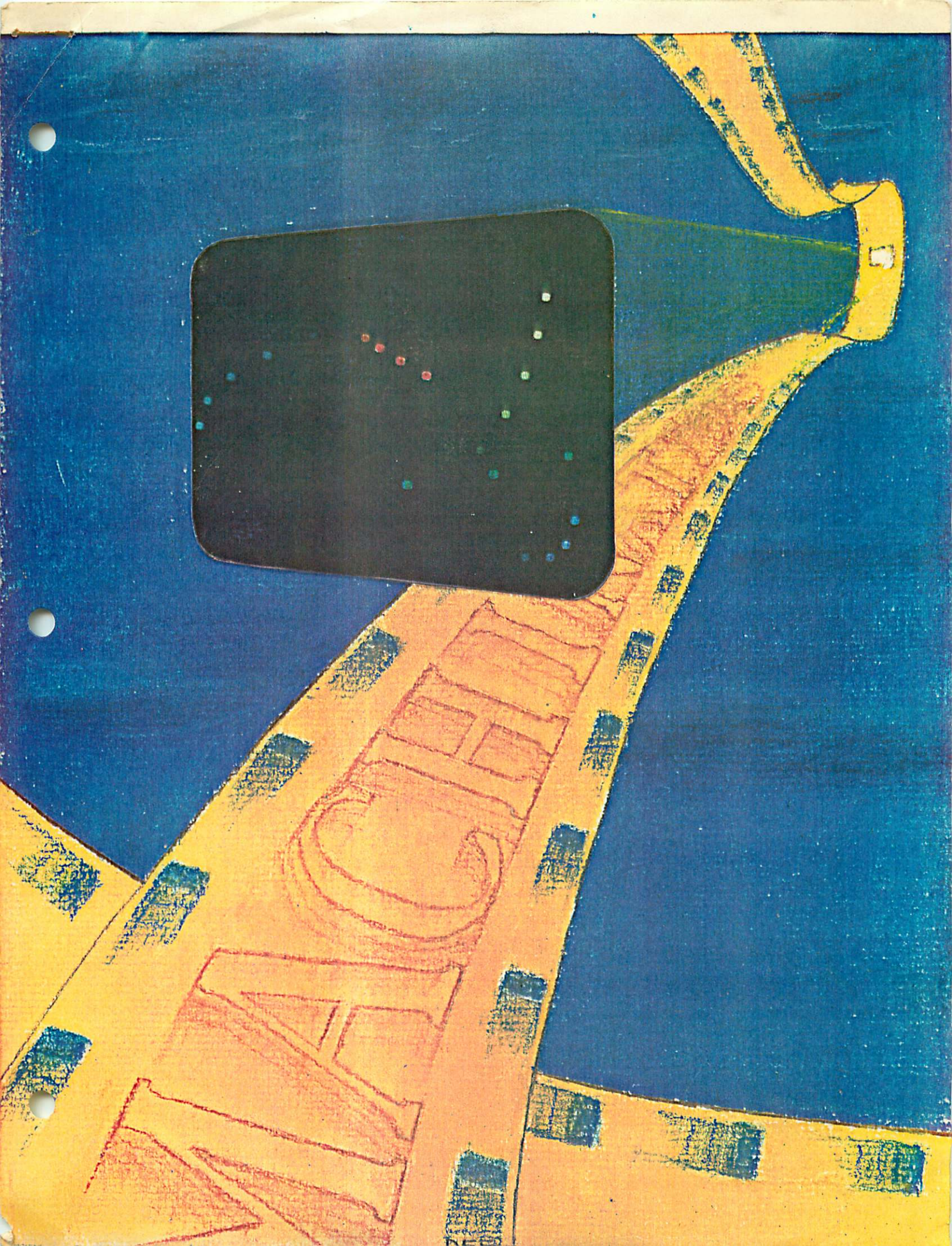


ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

Cover design by Dan Franzblau.

GUEST EDITOR'S NOTE by Paul Pangaro.

During the production of this week's issue, we found that there was too much material on our animation activities to fit in a single issue. Next week's Machinations will contain write ups on the LISP based system of Craig Reynolds; the current real-time experiments in sync-sound and color manipulations; and shape oriented graphics for raster scan. The combined two issues will be representative of the breadth and depth of activities at the lab in motion graphics.

IN THIS ISSUE:

THIS WEEK

Wednesday, June 28th	Chris Herot & Paul Pangaro go to Toronto to visit Gordon Pask.

AND BEYOND

July 7th	Nicholas returns.
July 8th	Nicholas & Dick Bolt visit with Craig Fields at ARPA.
	Nicholas & Chris visit John Lehmann at NSF.
July 18th & 19th	IBM site visit.
July 29th	ARPA site visit to see SDMS demo.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

VISUAL TECHNOLOGY AND THE THEATER

This week's guest contribution, written in the journalistic style, is by Louis Pangaro, M.D. Dr. Pangaro is Cheif Resident at Georgetown University Hospital. He has appeared on stage in roles such as Marlowe's Faustus, Macbeth, Shylock, and the Marquis de Sade.

In the not to distant future television will no longer exist, but there will be Holography. Right in your living room, you could flip on a micro-cassette and see your favorite stars "in person" doing the last tango in Paris. It will be the next step in visual entertainment.

Until recently, there hasn't been much change in this kind of entertainment, and in what we think of as theater, hardly any at all. The change from a thousand marble seats on the side of your downtown acropolis to a few hundred people in a wooden "O" was more a question of sociology rather than a great leap in building materials. When electricity became available for lighting, we simply got rid of the candles and kept staring out over the footlights into the dark.

Movies changed all this: it took the audiences out of the theater; or rather, it took out the actors and left the crowd staring at the screen, the action-packed screen. And it takes technology to manage action. It took the skill of the Romans to surround the stage of the Greek amphitheater with a ditch and a wall and stage their gladiators and animals; to flood the floor and stage flaming sea battles. The technology fit nicely with the demographic problem: occupy the masses immigrating to the big city - let them eat panem et circum. Junk food and the movies. Intolerance, Birth of a Nation, the motion of motion pictures - sacking cities, great battles - set the pattern early. It wasn't the morality of those pictures that made them successful, it was a whole world created up there in blazing action without a scretch of your imagination at all.

The theater keeps moving on its lonely way. A few productions here and there projects slides as a substitute for a painted set, often a suggestive atmospheric image or symbolic affair rather than a realistic movie locale. Like others we've used a Kodak Carrousal to change mood quickly or to throw up a rapid sequence of suggestive images so the audience wouldn't get too tired of imagining on their own. There has not really been too much change in that old actor-audience, both-of-us-are-here relationship.

Well, the great advantage to an actor or director in the movies is straightforward: the number of people who can see your work is numbered in the millions, not in the hundreds or thousands (and you can't hear any booing). Moveover, they can see your work as you left it, years later - even after you're dead. The advantage to the producer is even greater - he doesn't get the glory, he gets the cash. Four, five, six houses can be played a day and your actors don't even get tired - the blessings of a little technology.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

VISUAL TECHNOLOGY AND THE THEATER continued.

We all live intimately with television, pay it more attention than our spouses and thank it for watching the children. It has made entertainment and life easier for everyone. Viewers number in the tens of millions, a few percentage points in the ratings means thumbs up or down for a show. The great triumph, though, of TV is that we've been persuaded that the convenience is worth the advertising we have to sit through. Business has swiped the technology and chopped up experience into ten minute segments (five per hour). All this is not very taxing to the attention span, and each 30 or 60 minute show has no impact on the next hour's or next week's.

So far, there's been little use of television in the theater. Soap operas broadcast the live reading of a script but don't feature a living audience. The TV talk shows are the closest thing to a real merger of the two: camera watching stage and audience; audience watching stage and camera.

It will take holographic technique to get this into the proscenium. We'll start with the obvious ones: the ghost of Julius Caesar; the witches of Macbeth; "rent John Gielgud for the Ghost in your high school's production of Hamlet". This kind of thing will make the holograph useful and effective in the theater. The theater, grateful for funds, will reciprocate with a future series of Theater in America, bringing the Royal Shakespeare Company, the Boston Rep, and the Folger Theater Group into your livingroom in their latest Shakespeare or their latest Edward Bond. At first there will be trouble getting the scale for each scene right, but in the not too distant future, you'll see a Titus Andronicus ax off his own hand on your own carpet.

Immediate entertainment goals might be a little more modest. Let me share an idea for staging the first part of Goethe's Faust. The story of the scholar who sells his soul to the devil in return for youth and experience is most familiar in Goethe's version: half the play introduces Faust before his compact, introduces Mephistopheles, the demonic deprecator of human values, and culminates in the witch's kitchen with a fantastic scene with talking monkeys and smashed glass where the devil is enthroned and Faust rejuvenated. The second half of the play is about Faust's passion for the young Margaret. Seven consecutive and connected scenes are given to their growing relationship (with no diabolic or magical effects). Driven by his pact of restlessness he abandons her; she is confined to prison for murdering their illegitimate child. He attempts to rescue her but she is insane; as dawn breaks he escapes with the magic horses of Mephistopheles.

Staged in the early nineteenth century there was black powder for demonic tricks, costumed actors for the acrobatic monkeys, lots of make-up for the old witches. A modern staging should use what modern visual technology has to offer. More than just gimmicking up the effects, the implications of the technology are basic to the whole experience of the play in the theater.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

VISUAL TECHNOLOGY AND THE THEATER continued.

The opening scenes have Faust in his study, a dog follows him home from a brief walk and Mephistopheles steps out from behind the stove, i.e., the actor playing him steps out onto the stage. A scene later, a student talks with the devil whom he mistakes for Faust because of his long robe. In this scene our actor playing Mephisto speaks from behind a paper screen against which he is silhouetted by rear lighting. The next scene is in Auerbach's wine cellar. Our Mephisto does the magic tricks with wine and fire out of the table legs but all his lines of dialogue come from a tape recorder at the edge of the stage. And the pattern progresses in a like manner: we see Mephisto in a mirror as the actor playing him stands offstage or behind the audience; we see the actor playing Faust talking with the colorslide projections of the demon. The fantastic Walpurgisnacht episode is shown to the audience on film (Faust might be in this one) and is interrupted by the short, burlesques of the Walpurgisnacht Dream which are seen by the audience on closed circuit television (in style of gasoline and toilet paper advertisements). In contrast to this progressive technological distancing of Mephistopheles - which at the same time should be even more spectacular and funny to see - all the scenes with Margaret are done in person by the actress and actors on the stage in front of the audience. At the very end of the final scene as Faust abandons her and escapes with Mephistopheles, it should become apparent to the audience at the moment of their disappearance as the demon calls out, "Here to me!" that they were both holographic projections.

Well, I'm not sure when the technology for all this will be ready, much less of where we'll get the money to stage it. Eventually, some one will try to sell fresh bread in my livingroom with a holograph. As Rabbi Heshchel said: "The opposite of the human is not the inhuman - it is the diabolic."

Louis Pangaro
Washington, D.C.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM by Pangaro, Steinberg, Davis, and McCann.

ABSTRACT

EOM is an implementation of an interactive animation environment based on the following premises:

Simulation-based animation is the most appropriate kind for computers;

Scripting animation sequences with two-dimensional, spatially-arranged information on sheets is conceptually more advantageous than linear language typed at a console; and

Interactive and interpretive systems are esseantial for reasonable feedback for the animator.

This paper describes the appearance of the system to the animator and its basic internal organization; conclusions based on experience with the system point towards its successor system, Loom.

INTRODUCTION

Computer animation systems have been implemented in a wide variety of forms. Our work is derivative in the sense that it recognizes useful, conceptual perspectives explored by other systems. These include powerful modelling spaces (Smalltalk amd KRL); systems which move toward the blurring of animation script and animation sequence (Baecker -- Picture driven animation); and the fundamental point of view that animation activity is basically simulation activity (Kaye).

EOM (pronounced ee-oh-em) combines and expands these concepts, adding its own conceptual perspectives as described in this paper. Conceived as a limited experiment, this implementation incorporates necessary but confining compromises, including vector-based scripting and the distinction within the system of program and data. What follows is a presentation of the mannner in which the system is used to produce animation, the internal organization, and future directions. Loom, currently under design (and a direct outgrowth of EOM) will contain full power and be completely couched in a touch-sensitive, raster-scan display environment.

EOM has been the outgrowth of complex interaction over a period of a years time by a number of individuals. They are: Jim Davis, Larry DeMar, Dan Franzblau, Ben McCann, Paul Pangaro, Craig Reynolds, and Seth Steinberg.

ARCHITECTURE MACHINATIONS

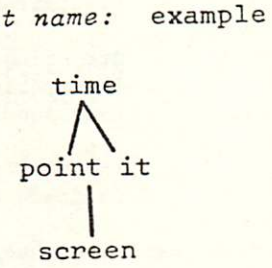
A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

SCRIPTING

The script editor is the basic environment for the animator. From menu lists presented on a dynamic vector-graphics display, primitives of the system called BIFs (Built-In Functions) are grabbed via lightpen and pulled unto the basic scripting structure called a sheet. A simple script sheet would be the adjacent figure.



The lines joining the BIFs are links which are specified by a minimum number of indications via lightpen, or button pushes on the keyboard with the free hand. Links are the simplest way of specifying syntatical relations on the sheet. Links are paths by which any variety of data type (such as a number, point, or shape) can be passed around the sheet from one node to another.

In Figure #1's example, the 'time' node has two links from its output, down which travel the global parameter of the current value of time, which increases each frame by some delta, specified at run time. The 'point it' node takes this data into its x input and its y input, and sends a point of that location down its output link. Multiple inputs and outputs are possible for a given node, but are not named on the sheet to prevent clutter. Sub menus appear during linking for lightpen indentification of specific inputs or outputs. The editor allows simple interrogation via lightpen of all the information relavant to links, nodes and sheets.

Note that the output of 'point it' goes explicitly to a 'screen' node, which causes it to be displayed when the sequence us "evaluated," as execution is called. Alternatively, the data could be sent to a 'console' node which would print out its value for debugging purposes, or into any other node which could take a point as meaningful input. The physical connecting and unconnecting, re-arranging, deleting, and intergrating of nodes inside the editor us extremely fast and easy to learn, and is one of the first advantages experienced by the animator.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

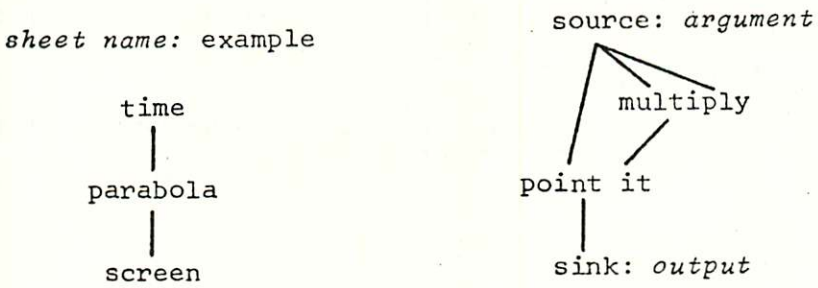
Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

This example produces a point moving linearly in time along the diagonal of the screen. To vary the path of the point, one could draw in an arbitrary shape and use the shape to define a path in time. The list of primitive transformations available in the BIF library is listed in a later section.

SUBROUTINING

In the context of simulations, what if the path desired were a falling parabola? One format might be:



This introduces the subroutining syntax, using a class of nodes called UDNs (User-Defined Nodes). 'Parabola' is defined inside the editor, using the nodes 'source' and 'sink' to pass arguments into and out of the sheet. These argument passing nodes contain names, speicified by the animator during linking, and these names are used during linking on the calling sheet to match the proper input or output. To invoke an instance on a calling sheet, the menu button 'user-defined' is hit via lightpen, and the name is typed at the console. An instance of that node appears, and is pulled unto the sheet and linked as if it were a BIF. No other action is required, with the system resolving all internal links.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

Only one sheet is viewed at a time, and to facilitate moving around the various levels of UDNs for complex sequences there exist menu buttons called DIVE and POP. Hitting the DIVE button and chosing a UDN causes the editor to display that particular UDN sheet, and make it available for editing. POP returns back to the calling level. Thus, the network fo subroutines can be traversed, diving and popping to any depth of the tree. This provides an interface consonant with the sheet concept, and in practice, becomes helpful in assimilating the structure of the conceptual script.

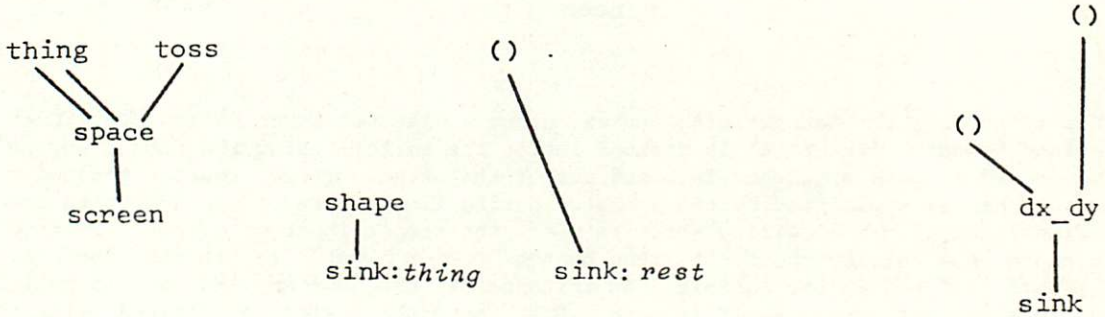
Subroutining is thus implemented in a completely clean manner, allowing for conceptual clarity while moving through the scripts, and requiring no management by the user to resolve links or keep track of status. Importantly, these aspects encourage a clarity of scripting since any conceptual element can be placed on any sheet as is appropriate for that sequence. These features are demonstrated in the next section.

A SIMPLE SIMULATION

Consider the following sequence, which simulates a 'thing' (an object) being 'tossed' into space, and viewed on the screen.

thing toss

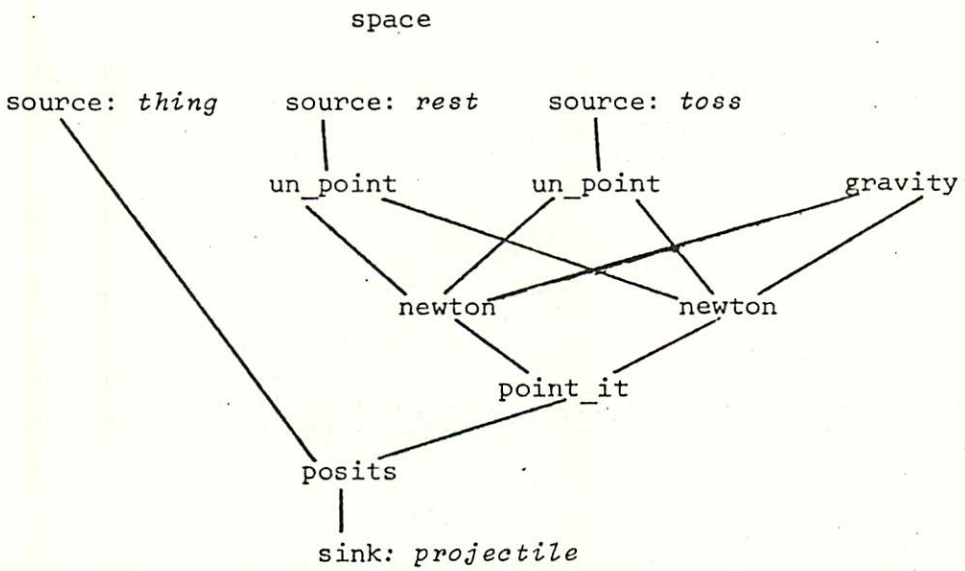
sheet name: throw

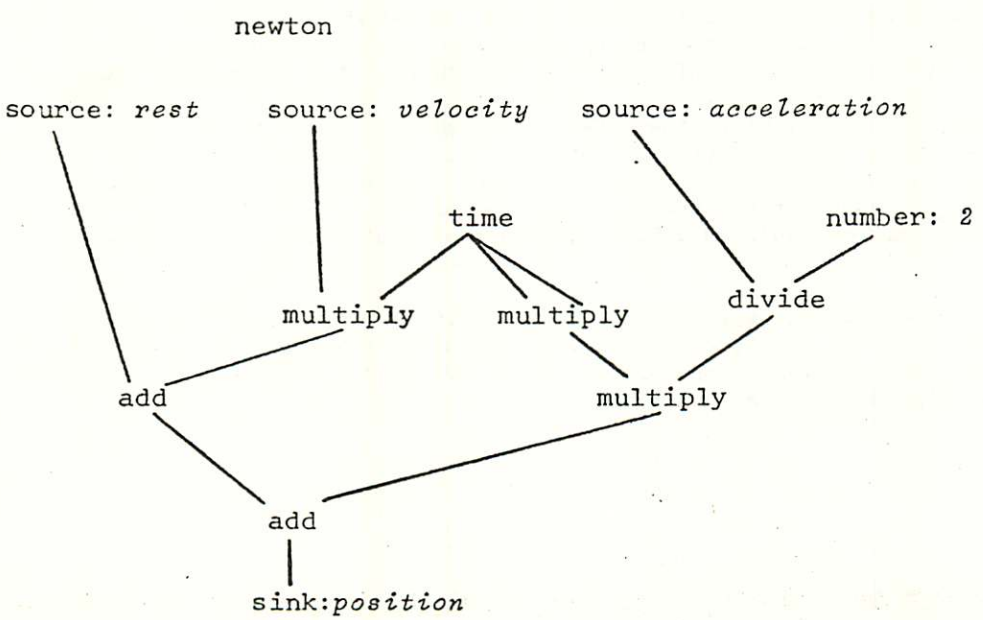


ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977





ARCHITECTURE MACHINATIONS

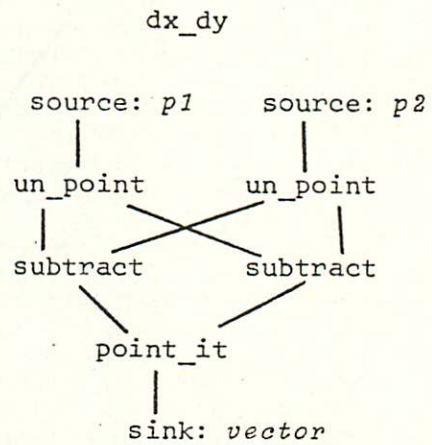
A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

On the top level, called 'throw', a 'toss' and a 'thing' are put into 'space', whose output is displayed on the 'screen'.

'Thing' is simply a defined shaped (a shape is a series of points which are to be joined by lines), and an initial position called 'rest'. The rest position is specified via the '()' node, which gives its absolute position on the sheet, as an x, y point. The coordinates of the sheet are identical to the coordinates of the screen, so placement of the rest position on the sheet 'thing' is determined by the position of the node'()' on the sheet itself. No concern for numbers is required, and the specification is completely graphical.



This extraction of graphical data from sheets is also used to specify vectors (magnitude and direction) in the 'toss' and 'gravity' nodes. The position of two '()' nodes on the sheet is passed into 'dx_dy', which computationally subtracts their x components from each other, and the same for y components. 'Dx_dy' returns this information in the format of a point data item.

'Space' takes its inputs, brings in the effect of 'gravity' (which is also defined graphically as a vector on the sheet), and invokes the UDN 'newton', which computes for x and y the position of the object as a function of time using the standard mechanics equation, $Sx = Sx0 + vxt + \frac{1}{2}ax\ t^2$. The lambda notation for equations is clumsy to read in EOM, and would advantageously be replaced by a simple algebraic interpreter.

The turnaround time inside the system for changing initial conditions, such as the vector of the toss or even the force of gravity, is just a few seconds. These parameter changes are achieved inside the editor simply by pointing at the node via lightpen and pulling the node to a new position.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

This uniformity between animation action (process) editing and initial values for parameters (configuration) editing is very pleasing. The value returned by the '()' node is in the current implementation a constant, but in later versions would be a variable, changing as the simulation moves the object's position. Thus, halting the simulation in the middle of the run would find the position of the node in the editor changed to the current position as specified by the simulation. In a uniform system based wholely in raster scan, the effect would be as follows: In the editor, an initial position for the shape is chosen. The menu command 'simulate' is invoked, and the network of nodes disappears and the actual shape moves from the rest position node up and across the screen. Halting execution at a particular frame results in automatic return to the editor, wherein the network reappears with the position node moved to the shape's position on the trajectory.

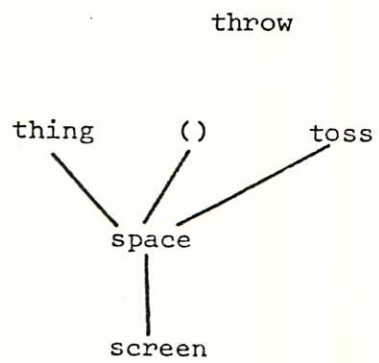
In less trivial examples, where the information extracted from the sheet is more than simply position (say, mass, or connectivity of neural elements) the feedback involved in the process (script) and product (animation) relationship is exceptionally powerful, and not available in any other animation system.

HOOPERS AND SCOOPERS

To facilitate the manipulation of nodes on a variety of sheets to produce a completely scripted animation sequence, certain editor functions are required.

A hooper is a function which allows the specification of a set of nodes to be manipulated together. Manipulations include moving a set of nodes as a cluster across the sheet to make the arrangement clear or meaningful; or to scoop.

A scooper is the function which allows the moving of a node or set nodes (a hoop) to a different sheet level, that is, to a different conceptual organization. Frequently while constructing a sequence on-line, nodes are initially placed inside a particular UDN inappropriately, as in this case:



ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

The rest position of 'thing' is on the top level sheet, and probably belongs conceptually (as a matter of taste alone, frequently) inside of 'thing'. By indicating the node '()' as a hooped set, and the UDN 'thing' as the place to embed the hoop, the action scoop is invoked by lightpen. This places the node'()' inside of 'space', automatically resolving links, modifying internal argument lists, and adding the necessary 'source' node inside of 'space'.

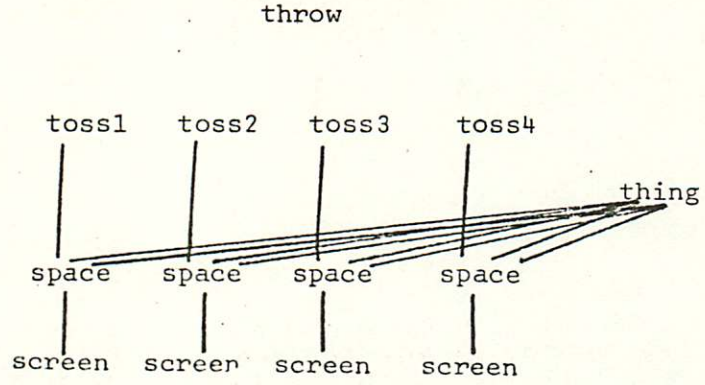
The system will also resolve editing actions such as the modification of subroutine input/output lists even while higher level invocations of that routine are rendered inconsistant by the editing. For example, when an input is deleted from a UDN, sheets which call that UDN will automatically resolve those links which it can, and signal the user of places where further editing is needed.

These editing functions are extremely important because they encourage experimentation with conceptual organization of scripts. It has been found that the sheet syntax, combined with the use of the conceptually simple parcelling up of functions and activities of a simulation which produce clear but numerous UDNs of evocative names, leads to clarity of expression in the animation, and is a major attribute and innovation of EOM.

SERIAL/PARALLEL

Quite often, multiple instances of objects are to be displayed in a sequence. To minimize the number of instances of nodes necessary to define a sequence, and to keep the conceptual organization of a script as clear as possible EOM contains the facility for parcelling data items. At any point in the scripting, multiple instances of any type of node can be avoided (if desired).

The parallel function is implemented in the form of a 'group' node. This node takes a number of inputs, and internally strings the data into a set called a group which it passes down its output link. Thus, for example, the following two sheets are equivalant in their painting of the screen:



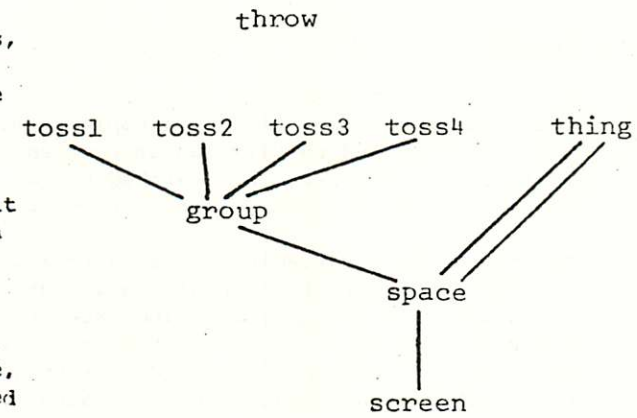
ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

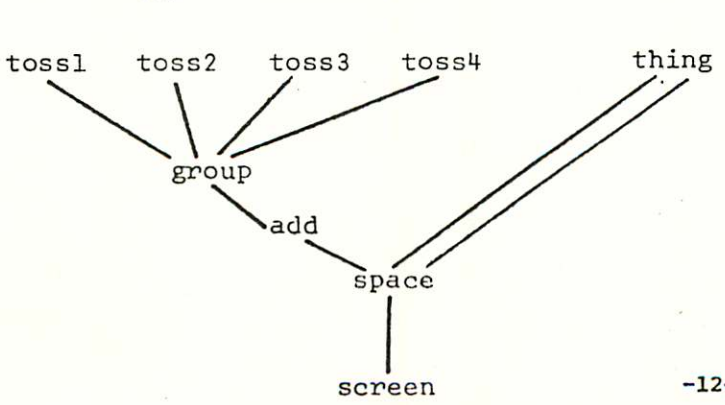
EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

The group in the adjacent figure passes each vector from each toss, in turn, to the 'space' node which maps its other non-group input arguments per invocation. After evaluation, these are grouped again and passed into 'screen'. Inside the 'space' node, each downward link passes the group of three data items, which at each node are individually mapped as before, and so on down the tree.



If 'thing' were replaced by a group of three different things, the result would be three objects each thrown a different way. Thus, if more than one input to a node is a group, the elements of each group are matched, and mapped unto the node one by one. Groups may be of arbitrary structure, that is, groups of groups are handled consistently. Any data item of the system may be grouped at any level in the network which comprises the complete script, and all subsequent operations below that point in the network operate on the individual elements of the group. All these effects are controlled by the interpretor, which results in simple coding for the BIF library. This is discussed further under the section on internal organization.

The conceptual inverse of the 'group' node is a serialized function. For ease of implementation in the current EOM, the decision was made to use serial input ports to the nodes, rather than have an explicit 'serialize' node. Thus, in the thrown example, to pass the sum of the vectors into the space, the sheet would be configured:



ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

The output of the 'group' node is a group, which connects to the serial input of the 'add' node. This is visually implied by the 'side' link to the node, and is achieved in the editor by a menu button. The output from the 'add' is a single element, the sum of the serial inputs. The use of the parcelling relieves the need for multiple instances of 'add', and in practice, can reduce the number of nodes greatly, while also contributing to conceptually clarity. There is a later example under the section on entity simulation of this.

MEMORY

The addition of memory to the system is necessary for a wide calss of modelling to take place. Because of the multiple sheet leveling of the scripting, a simple syntax had to be devised to distinguish between global parameters and local instances. All variables in the sytem are matched by name, consist of an initial value set in the editor, and have either inputs, outputs, or both.

The output of a variable evaluates to its present value, straightforwardly. The input link to a variable contains a variety of information about the particular usage. The absence of an input link indicates that the value of the variable is defined somewhere above (that is, back up the stack of invoked UDNs) and not in the current sheet; the stack is therefore searched back until the name match is found, and the value read. This is an implicit global value search.

If an input is present, the variable is tagged as a local instance to that sheet, though UDNs invoked from (below) that sheet may access its value by name, as just described. If the value of the input link is null (boolean false), then the value of the variable is not changed during this frame. If a value is present, then the value is updated at the end of the frame, so that during the next frame this value is available at the output of the variable. In practice, this scheme is quite sensible and intuitive, and allows a flexibility of memory organization which is clear. Ring-buffers, flip-flops, sequencers, and echo-images are easily implemented. The next section shows examples.

ARCHITECTURE MACHINATIONS

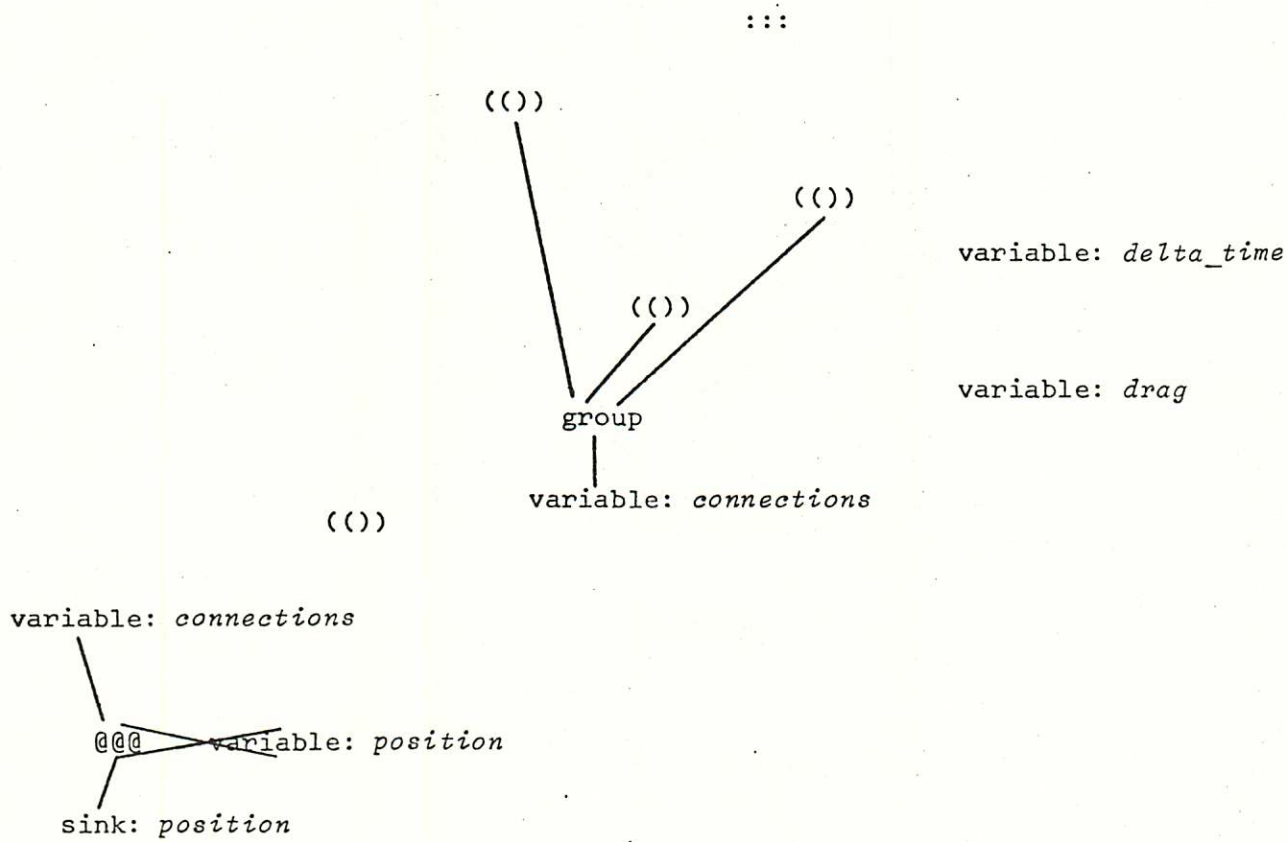
A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

ENTITY SIMULATION

Two-dimensional scripting suggests an easy way to explore a wide variety of simulation classes, including automata evolution (ref. - see abstract in this issue). The folowing favorite class of entities behave as if they were attached to one another by rubber bands.

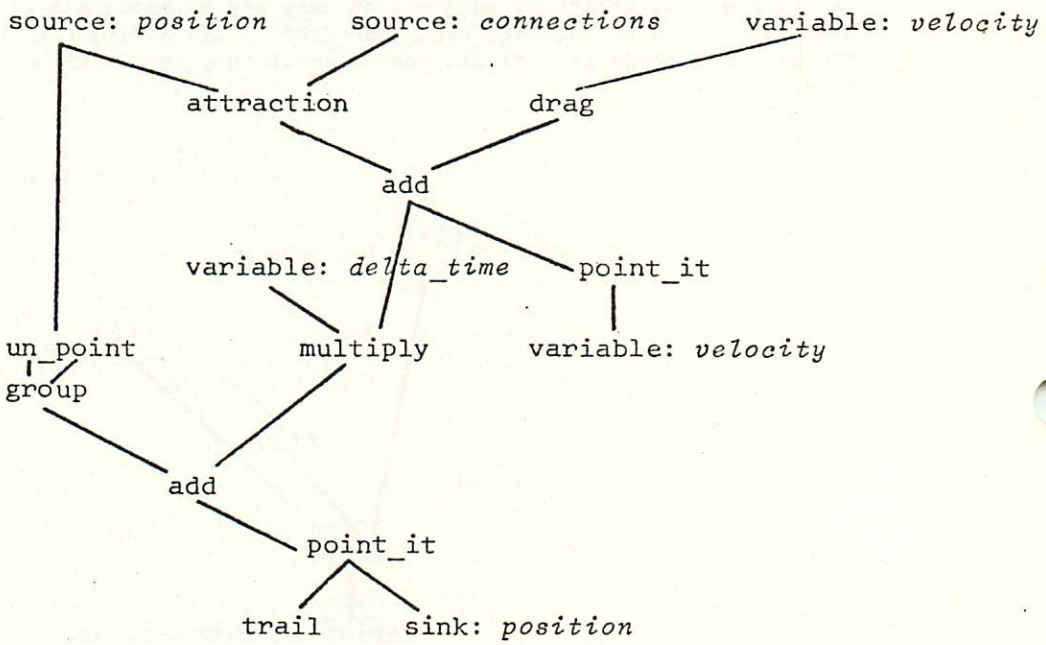


ARCHITECTURE MACHINATIONS

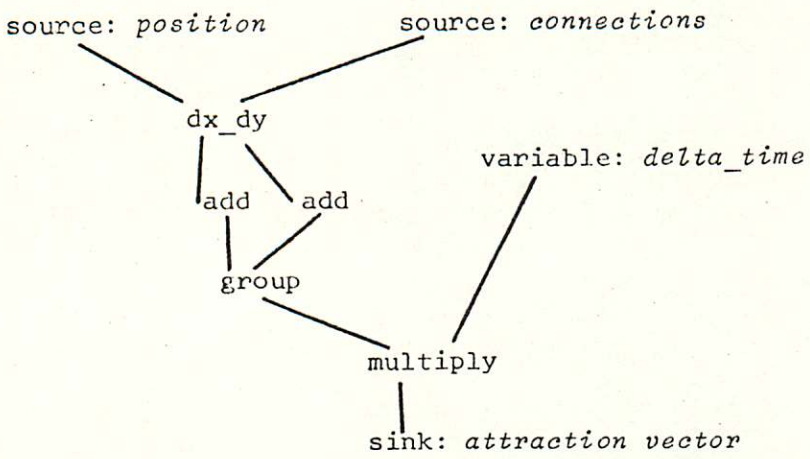
A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

@@@



attraction

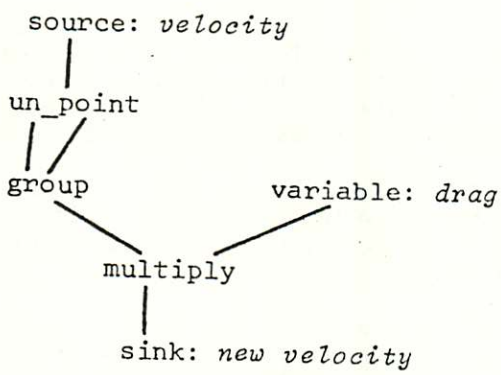


ARCHITECTURE MACHINATIONS

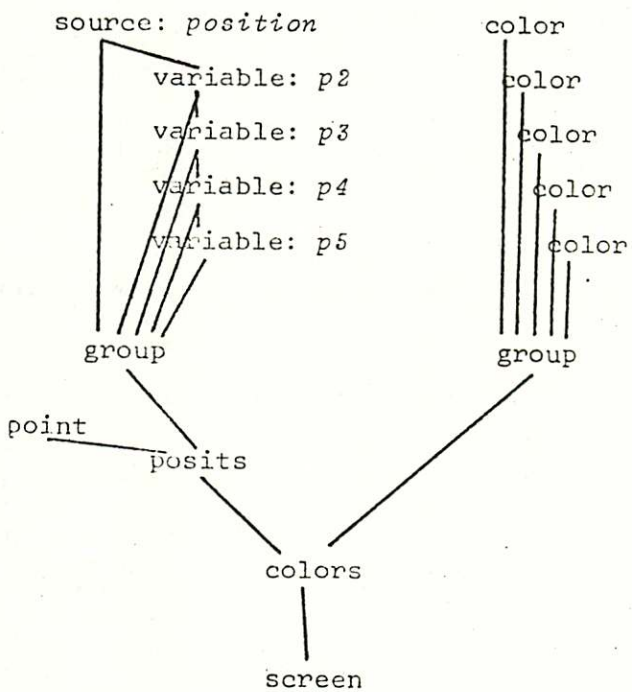
A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

drag



trail



The entities '(())' each output their position, which in this example are grouped and stored in a variable called connections. Also on top level (called: ':::') are stored the global variables delta_time and drag, which are accessed deeper inside the net.

On the '(())' sheet is stored the variable position, which for each instance of '(())' on the sheet ':::' stores the current position. This position and the connections are passed into '@@@' which does all the motion computing. On '@@@', the current position of this instance of '(())' and the group of positions of the other entities pass into 'attraction'. On that sheet, the position of this instance (position source) is compared to all other instances (connections source), and a group of delta distances is passed out of 'dx_dy'. (Note that the

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMUALTION-BASED ANIMATION SYSTEM continued.

'dx_dy' operation is performed once with both arguments being the same, since the instance position is contained in the group of connections. The result is zero, and computationally falls out.)

The output of 'dx_dy' is a group of x differences and y differences, which are each summed by the serial add. The sum of x and the sum of y is made parallel by the group node, and each is multiplied in turn by the delta_time variable, which determines the time slice. The resulting group of x,y attraction is passed out the sink node.

'Drag' on the sheet '@@@' takes the current velocity as argument. The velocity, in point format, is made into parallel format by the 'un_point' and 'group' nodes. The x and y velocity is then multiplied by the drag factor, which means that drag is a function of speed. This is put out as a group of x,y.

On the '@@@' level, the results of the 'attraction' and 'drag' effects of velocity are summed in the add node. Since this data is in parallel form, it is passed into the serial input of 'point_it', which outputs a point (interpreted always as an x and y vector of velocity), stored in the velocity variable. Note that there are two instances of variable: *velocity* on the sheet. They indicate the same memory cell, and their inputs indicate that the new value is set at this point, as a result of the computation just described. The instance at the top of the sheet, which has only an output connection, provides the current value (this is discussed more explicitly in the section on variables). In the meantime, the position point passed into the source at the top of the sheet '@@@' is transformed from a point into a parallel x,y format by the 'un_point' and 'group' nodes. This necessity will be avoided in later implementations by more sensible internal formats for the data. The result of this transform combines with the new velocity in the 'add' node, resulting in the new position. Before the velocity is combined with the position, it is multiplied by delta_time to compute velocity times delta_time equals delta position. Still in parallel format, the x,y data is connected to the serial 'point_it' node. This is returned to '(())' where it updates the position variable and is sent up yet another level to ':::' where it is available for the next frame to provide the updated positions for the connections variable.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

The computed position for this instance is also passed to 'trail' from the '@@@' sheet. This UDN displays the object, which consists of a series of points at its previous n positions where n is the number of cells in the shift register implemented by the chain of variables, as shown.

Each time step, the new postion shifts down into the register, variable by variable. The outputs at each time are grouped and passed into the positioner node 'posits'. As explained under the section on serial/parallel, this results in a group of five points returned from the 'posits' node. These five are matched, one by one, to the group of five colors when the 'colors' node is invoked. The result to the screen is five points, at the current and previous four positions, each a different color. This resulting 'trail' is shown in the sample frame on the cover of this issue.

Though there is only a single instance of 'trail' in the entire script network, its mutliple invocations from top level by the '(())' nodes cause the proper instancing of the shift register to occur. Similarly for the velocity and postion variables. Note in ':::' that the two variables drag and delta_time simply sit, their initial values being set in the editor and never changed. They are accessed inside of every invocation of '@@@' and 'attraction' and 'drag' with no user special actions.

This example shows in detail the various workings of the system's features. The variable instancing scheme is extremely easy to adopt, and makes implementing complex interacting processes simple. Note that the explanation of any set of UDNs is more difficult than the creation of them inside the system. The variable scheme, the UDN sheeting, and the various editor functions provide an environment that is both a pleasure to use and is conceptually transparent and helpful.

ATTRIBUTES

The limitation of variable usage just described is that information can only be accessed below the instance of a particular variable. Frequently, it is useful to pass the information across the network, particularly in entity simulations. This is particularly true when the modelled environment is uniform, that is, all entities interact with all instances of entities on the sheet. Eliminating the need for links from all entities to all others is conceptually better and makes for practical scripting. The following resolution of this weakness, though not implemented, is consistent with the current system.

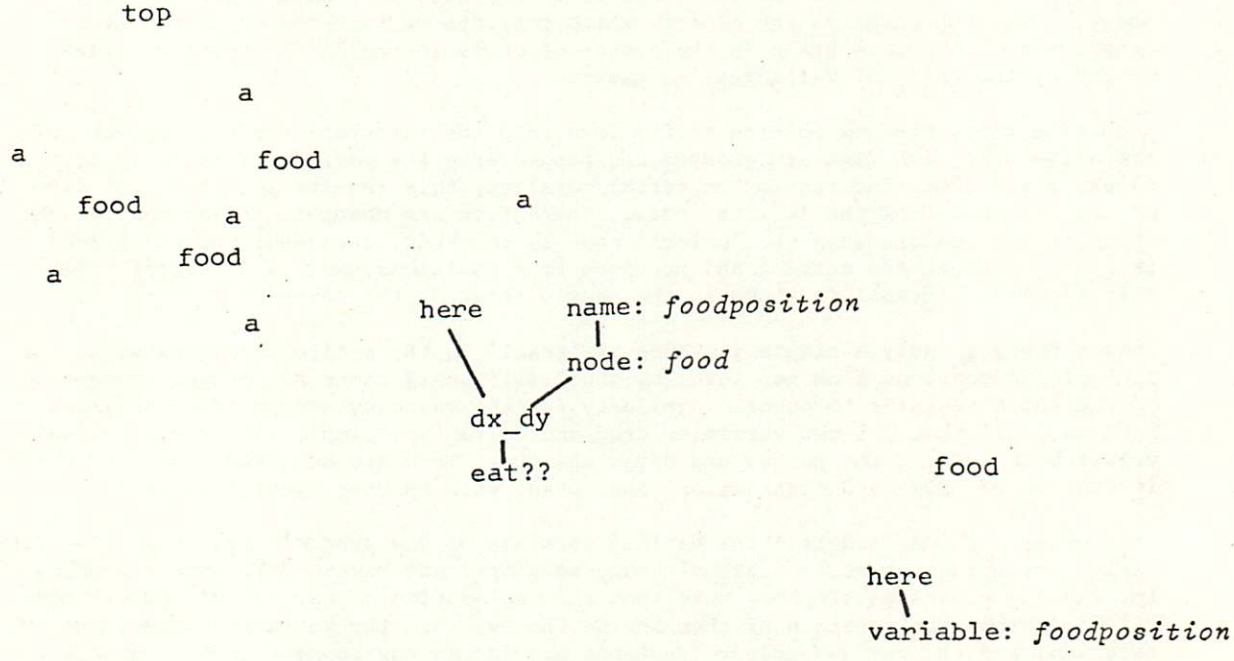
ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

Consider this hypothetical script:



The top level contains a few entities, say, cellular automata, which interact as a function of distance on the sheet. Each entity needs position information of all its neighbors to 'see' if closeness relations allow for eating of food or the forming of coalitions. This is achieved as shown, by using the node 'name' to indicate the name of the variable in the sheet 'food' to be extracted. When encountered by the interpreter, these nodes cause the search of the database for the variable 'foodposition' in all instances of the node 'food'. These are, one by one, passed into the node 'dx_dy' on sheet 'a' to determine if that food morsel can be eaten, or whatever.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

This is expressly the all condition, since the case of 'a' being related only to specific instances of 'food' is included in the case above with'(())', wherein specific connections are made to indicate the instancing. Also, the extraction is made across on the current level in the network as a first restriction of implementation, since it is not clear what the implications of multi-level extraction of attributes in the network are.

INTERNAL ORGANIZATION

The keystone of the EOM system (implemented in PL/I) is its interpretive executor of the network, called the evaluator. The evaluator operates on the database which is constructed by the editor under the command of the animator. Invoking instances of nodes from menus, linking, setting constants, and creating sheets all modify the database which defines the particular script.

The current implementation of EOM maintains a distinction between program and data, and hence the two basic formats are items and nodes.

Items

Items are database elements such as numbers, booleans, colors, points or shapes, which are passed down links from node to node as arguments, and ultimately, in the case of points and shapes, to the 'screen' for display. Numbers consist of float values; points have x and y float values; colors are specified by intensity, hue, and saturation values; shapes are points to be connected by lines by the 'screen' node during evaluation; etc. These items are operated on by the nodes themselves.

Nodes

Nodes are either BIFs (built-in functions, taken from the editor menu lists) or UDNs (User-Defined Subroutine Nodes, which can be called from disk or defined on the fly). Nodes either transform data items from one format to another (such as 'point it' which takes two numbers and outputs a point) or transforms the data (such as rotate, which takes a shape and an angle, and rotates the points of the shapes).

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No 23 June 28, 1977

EOM: GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

The internal database of a BIF consists of a data block for each instance of the node and a description block which is shared by all instances of the node. The node instance contains pointers to the nodes which are inputs and outputs, flags, and a pointer to the description block. The description block contains information about the number of input and output arguments; argument type information for error checking in the executor; and the code entry point in the BIF library.

User defined nodes have a similar description block, but instead of an entry point to code, the UDN description block has a pointer to a linked list of node instances which make up that UDN.

EVALUATOR

The evaluator walks this list of nodes looking for a node which has all of its arguments (if any) evaluated. Once found, it checks the argument types and then executes a subroutine call to the appropriate BIF code. If the node is an instance of a UDN, the executor calls itself recursively. This process is repeated until every node in the network has been evaluated once. This is done every frame. An auxiliary prepass can be made to order the nodes in the network to minimize the number of passes over the list in search of nodes whose inputs have been evaluated. Once evaluated, any node has a value which can be any item type.

GROUPS

Items can also be linked into sets called groups. Groups are created by inputing a number of instances of items (that is, outputs of nodes) into a 'group' node. The output of the 'group' node is the group whose elements are its inputs. The executor breaks up argument sets (n-ary trees) by walking the fringe of the tree and passing the individual 'leaves' to the BIF. If only one of the input arguments to a node is a group, then the other arguments are used repeatedly and once for each leaf and a new group is constructed from the results of the group of BIF calls. If more than one argument is a group, the fringes of all group arguments are matched, one for one, and the same process is repeated until the smallest group is exhausted.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 23, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

MULTIPLE INPUT/OUTPUT ARGUMENTS

Nodes (both BIFs and UDNs) may have multiple inputs/outputs. These are handled by building a list of the individual outputs with each referencing node knowing which of n outputs it wants.

SERIAL INPUT PORTS

Several BIFs have one serial argument which can be used instead of the normal input argument set. The argument to the serial input port must be a group. The evaluator breaks this group up in one of three ways:

1) It can accumulate the results of the BIF function as it walks the group leaves. For example, 'add' will sum the elements of the group;
2) The evaluator will parse the group into subsets which are mapped onto the arguments of the BIF, which is then called. This is repeated until the group is exhausted. The final result is a reduced group of values. For example, passing the group (1 2 3 4) to the serial input of the BIF 'point it' would result in the group of two elements which are points: (1,2 3,4); or
3) The group is passed intact for special casing inside the BIF.

VARIABLES

Variables are bound in each UDN in which a link passes into the named variable. If no input is indicated, the variable is a reference to the nearest binding of that variable name on the execution stack of UDNs. Each instance (call) of a UDN which binds a variable has a memory cell for that variable in that instance of the UDN. Variables are pre-initialized in the editor and their new values are propagated at the end of each frame. The value of a variable is always defined. Because of this and the delayed propagation, first-in-first-out and circular buffers can be made simply from chains of variables. Circular networks can be made provided there is at least one variable in the circle.

CONDITIONALS

The system has conditonal constructs which either 'switch' one of the two inputs according to a boolean truth value, or 'gate' one input according to same. If a 'gate' is closed, all succeeding nodes will be ignored with the exception of variables. Variables, instead of changing their value that frame will retain their old value until the 'gate' opens.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

LIST OF PRIMITIVES

The following list consists of those BIFs currently implemented. The set was determined by arbitrary choice, and limited to core size. The coding of BIFs is simple, since the evaluator performs all overheading, such as subroutine calls, group operations, and variable binding.

NAME	INPUTS	OUTPUTS	ACTION
Constants, values are specified in the editor:			
number	---	float value	
point	---	x and y float values	
color	---	intensity, hue, and saturation values	
shape	---	flood point, color, and line definittions	
()	---	:null data & position (see 'here' node below)	
here	---	point	absolute postion of the instance of the called UDN on the invoking sheet.
Controls:			
console	data	---	prints out input data
screen	data	----	draws display data on screen
time	---	global time	
sink	data	----	passes data to invoking UDN
source	---	data	extracts data from invoking UDN
variable	(new value)	(current value)	memory (see section on variables)
slate	color	----	sets background color

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

LIST OF PRIMITIVES continued.

NAME	INPUTS	OUTPUTS	ACTION
Arithmetic:			
* + ÷ -	a	result	performs operation on inputs
	b		
**	base	result	exponentiation
	power		
random	---	number	output a random number
Item Makers, these re-format data items:			
point it	x and y	point (x,y)	
color it	i value	color (i,h,s)	
	h value		
	s value		
shape it	flood point	shape(...), with lines drawn between points	
	color		
	points		
Number Crunchers:			
sin	angle	sin (angle)	
cos	angle	cos (angle)	
modulo	a	mod (a,b)	
	b		

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III. No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

LIST OF PRIMITIVES continued.

NAME	INPUTS	OUTPUTS	ACTION
Shape Manipulation:			
interp	shape percent	point	interpolates along the points of the shape and returns a position on the shape at percent distance along the curve.
posits	shape & (scale) & position	shape	moves a shape to new origin at position; scale performs an x,y scaling of the position
colors	shape & color	shape	changes the shpae color to new value
rotate	shape & angle	shape	rotates the shape by angle
scale	shape & factor	shape	changes the size of the shape by factor
modify	shape & data & which	shape	replaces the data in the shape by which to the value of the new data: if which < 0, then change flood pt; if which =0, then change color; if which >0, then replace nth pt.
Groups:			
group	data data : :	group	makes a group of the input elements outputs it
series	from to incr	group	performs a do-loop on the inputs, and outputs all values in the series as a group

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

LIST OF PRIMITIVES continued.

NAME	INPUTS	OUTPUTS	ACTION
Conditions:			
gate	data boolean	data	if boolean is true, then outputs data; else outputs null
switch	boolean data 1 data 2	data	if boolean is true, then outputs data 1; else outputs data 2
greater lesser equal	arg a arg b	boolean	outputs a boolean based on results of comparison operation
Extractors, these are the reverse operation of item makers:			
un point	point	x y	
un color	color	i h s	
un shape	shape which	data	for which >0, outputs the nth point; for which =0. outputs the color; for which <0, outputs flood point.
node pos	any node	position	outputs the absolute position of the argument node on the sheet.

ARCHITECTURE MACHINATIONS

A weekly newsletter of the Architecture Machine Group, Department of Architecture, M.I.T., Room 9-518, Trilla Ramage, editor.

Vol. III, No. 23 June 28, 1977

EOM: A GRAPHICALLY-SCRIPTED, SIMULATION-BASED ANIMATION SYSTEM continued.

SUMMARY

EOM has proven to be a fertile testbed for a variety of assumptions about the nature of graphical animation on computers. The descriptions in this paper are barely evocative of the ease with which wide classes of animation sequences can be implemented quickly.

Further, the sheet approach has created an environment for exploring conceptual programming in which the relation between mental concept and physical script has some meaning for the animator. The levelling of subroutines, graphical information extraction from 2-D scripts, and dynamic data configuration within the networks are all mutually contributory to a powerful conceptual system.

This work precurses systems which completely blur the distinction between programming and animating. Environments in which programming is performed by creating animated sequences, rather than vice versa, are possible. Loom, the next generation of EOM, will contain a full implementation of all the features implemented and suggested here, fully based in raster-scan hardware. With the programming space (the editor) and the animation space (the moving sequence) further merged unto the same display area and viewed under a transparent, touch-sensitive tablet, a major stride toward purely visual programming will be made.