6-1989

ESCIM: A System for the Investigation of Meaningful Motion

Siew Hong Yang
University of New Brunswick

Colin Ware
University of New Hampshire, Durham, colin.ware@unh.edu

ESCIM: A System for the Investigation of Meaningful Motion

Siew Hong Yang
Colin Ware
School of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, New Brunswick
CANADA E3B 5A3
bitnet: cware@UNB.ca

Abstract

A language is described whose purpose is the investigation of meaningful motion using Stimulus Response animation techniques. The language is capable of adjusting the shape, size and velocity of an actor in real-time computer animation. Some results are presented showing how it is possible to generate such behaviours as chasing, avoidance and hitting using this animation technique. A set of primitives are presented which we find invaluable in the control of size, stretch and velocity parameters when attempting to produce fluid and meaningful interactions.

Keywords: Motion Control, Animation.

Introduction

Stimulus Response (S-R) animation is computer animation achieved by giving objects in a simulated environment rudimentary sensory capabilities and the ability to respond to other objects in the environment. Responses typically take the form of movement or shape change which secify the state of the objects in the next display frame. This simulated environment is placed in a loop, each cycle of which generates images based on the previous states. By this mechanism the environment evolves over time and its creators can observe the "social" interactions which take place. (Willhelms, 1987; Lethbridge and Ware, 1987, Marion, et. al. 1984).

One of the most interesting challenges for S-R animation is the discovery of simple rules for synthesizing purposeful behaviour, or more precisely, for creating the impression of purposeful behaviour (we do not wish to credit our programs with intentionality, only with the appearance of intentionality). This would seem to be a difficult task but for the work of psychologists such as Michotte(1963) and Basili(1976) who have shown that it is possible to induce such apparently complex percepts as hitting, pushing, and aggression with remarkably simple motions. The brain, apparently, is quite ready to attribute purposeful intentions to simple geometric objects given certain simple motion paths. There is also the work of "synthetic psychology" (Braitenberg, 1984) which appears to show that high level concepts such as love and hate may be more easily synthesized than deduced by an analysis of human (or animal) behaviour.

The ESCIM (Expressive Shape Control In Motion) is a software system designed to make it easy to test the effect of simple S-R rules on perceived behaviour. The system is a direct successor to the PAM (Perceived Animate Motion) system (Lethbridge and Ware 1987; Lethbridge and Ware, in press) which we briefly describe first. Like ESCIM, PAM is implemented on a Silicon Graphics IRIS Workstation. PAM is a low-level *T2* S-R control system which means that PAM generates sequence of animation frames by making the responses of its actors a function of the previous two environment frames. In contrast to high-level systems, actors in PAM respond directly by changes in position to changes in their environment. PAM is also a two dimensional system capable of controlling only planar environments. Pam's language includes structures called Partial Response Functions which define the response tendencies of an actor caused by the states of its stimulus objects and also itself. The values returned by Partial Response functions are combined (by weighted summation) to produce the actual Behaviour Functions. A disadvantage of PAM's language is that it prohibits partial responses which involve a relation with more than one stimulus object. Thus the user cannot tell a goalie to stay between the goal and the ball, for example in a simulated hockey game. Although the partial responses to the ball and to the goal can be summed to produce something which looks similar. Despite such limitations PAM achieved its goal which was to demonstrate that with simple deterministic behaviour functions, a rich and interesting set of behaviours such as pulling, pushing, escaping, and chasing can be produced. Moreover, given the existence of the correct primitives, it can be relatively easy to produce smooth natural motion.,

Building on the success of the PAM system, ESCIM is a new S-R animation system that provides better extensibility and flexibility, that uses a more readable language, that allows responses in the form of shape change and colour change as well as position change, and that works in a 3-D space. Like it's predecessor, ESCIM also only allows responses based on two previous time frames (T2), but unlike

PAM, ESCIM abandons the idea of a stricly heirarchical Behaviour funcition in favour of a more general language based on a rich library of stimulus and response functions.

Software Overview

The basic goal of ESCIM is to allow the rapid prototyping of S-R animations. As such, it is of paramount importance that it be capable of producing animation in real-time. The idea is to investigate meaningful motion, not the animation of complex rendered objects. To achieve this rapid prototyping of animations we used a two stage approach. First the animation script, written in the ESCIM language, is complied into a linked list of trees containing pointers to functions. At the second interpretive stage this structure is traversed to produce the real time animation. If, because of especially complex S-R functions, the animation is still slow, then the output of the interpreter can be sent to a file as a set of graphics primitives. This file can subsequently be "played back" by a simple program which reads the graphics commands and executes the appropriate graphics primitives. Thus, the software system of ESCIM consists of a compiler, a function library, a display interpreter and a postdisplay processor.

Developed in the UNIX programming environment, the parser component of the compiler has been generated using the *yacc* software development utility. Given an input script describing the desired animation scenes, this parser produces a linked hybrid-node structure which is formed from two different types of nodes : *conditional* and *response function.* A conditional node can contain a conditional expression which incorporates a pointer to a stimulus function (to be evaluated at run time). The conditional node also, has an **if** and an **else** pointer that can point to another conditional node (and hence allows nested **if-then-else** structure in the system) or to a response function node. A response function node contains a pointer to one of the response functions in the function library and it can be linked to another node of the same type or to a conditional node. Other components of the ESCIM compiler are a a scanner and an error handler. The scanner generates a listing file which may contain error messages and warnings from the error handler.

Once the input script is parsed and checked to be error free, and the linked hybrid-node structure is created, the display interpreter is called to traverse the linked structure and hence to invoke functions in the function library. The function library contains both the stimulus and the response functions that are used for computations and for performing responses; these functions define the sensing and responding abilities of the actors. The response functions act directly on the stored parameter values (velocity, size, etc.) of whatever objects are specified. Whether a given response function is called or not depends on values returned by the stimulus function, (or functions) as specified by the ESCIM script. Once the end of the linked list is encountered, the display interpreter uses the modified object parameters and invokes IRIS graphics routines to generate the new graphic display frame. Following this a time frame is said to be elapsed and the whole linked node structure is re-traversed.

Despite the emphasis on real-time performance, ESCIM will slow down if either many objects or complex S-R functions are envoked. For these occasions the output of the real-time animation can be saved in a file of graphics primitives subsequently played back. Since during playback no calculation of S-R functions is needed the result is a considerable speedup. An additional advantage of operating in this mode, is that as the frames are saved, a depth sorting of objects is done. This causes objects to be displayed in the correct order for hidden object removal (this is not done in the normal real-time display).

The Language

The ESCIM language is free-format and structural; semicolons are used as statement separators. An ESCIM input script contains the following blocks, with those blocks embraced by '<' and '>' being optional :

< color definitions >

< environment controls >

< static object declarations >

(actors declarations)

< initializations >

(behaviours)

Optional color definitions are defined using the following syntax :

define <color name> <color map index>

Environment controls provided in the ESCIM system are as follows :

Environment Control	*Function*
def_viewport	defines screen area for displaying the image
def_perspect	defines a perspective projection
def_ortho	defines an orthogonal projection
def_lookat	defines a line of sight and twist angle
def_polar	defines viewer position in polar coordinates
def_color	initializes a color map color
def_mouseobj	attaches the 6-D mouse to an object

These optional control functions are invoked in the same way as functions in C and defaults are used if not provided.

The reserved words "static" and "actor" are used to begin static object and actor blocks. Two types of objects are currently defined in the ESCIM system, namely cube and ellipsoid. There is only a single version of the ellipsoid, ren-

dered using a series of filled circles. However, a cube can also be filled, open at the front or unfilled. Reserved words "ellipsoid", "fabox", "fbox" and "box" are used to declare an ellipsoid, a filled cube, a front-opened cube and a unfilled cube respectively along with nine parameters : position in 3-D (3 parameters), orientation (3 parameters), stretch factors (2 parameters), and size (1 parameter).

The optional initialization block provides a way of giving each actor a starting velocity. This block shares the response functions of the behaviour block. The word "init" is reserved to identify the block.

Behaviour Definition

The "behave" reserved word starts the behaviour block. Two types of statement are defined: (1) an **if-then-else** conditional structure, and (2) a response function call. The **if-then-else** structure is a follows:

```
<stmt> ::= if <cond expr> then
               <stmt>
           else
               <stmt>
           endif;
         | if <cond expr> then
               <stmt>
           endif;
         | <response fcn name> (<parameters>);
```

Parentheses are optional between the "if" and "then" reserved words. Four kinds of conditional expressions are defined. Note that these have stimulus functions embedded in them.

```
<cond expr>::= <stim fcn> <rel op> <scalar>
             | <stim fcn> <rel op> <stim fcn>
             | < scalar> < rel op> <stim fcn>
             | <cond expr> <bool op> <cond expr>
```

The relational operators are '<', '<=', '>', '>=', '=' and '<>', while the boolean operators are '&' and '|' for logical and and or respectively. The system is designed to make it easy to add new functions to the environment. Some of the stimulus and response functions which we have installed so far are listed in the tables below.

Stimulus Function	*Function Description*
timetoreach	calculate the number of time frames for one object to reach another
distance	calculate the nearest distance between two objects
changedist	calculate the changed distance between two objects
velocity	return the current velocity of an object
volume	return the volume of an object
size	return the major axis length of an object

Response Function	*Function Description*
addspeed	increase or decrease the magnitude of velocity
against	cause one object to move in opposite direction to the other
gravity	cause an object to be under the influence of gravity
invisible	cause the image of an object not to be displayed
away/towards	cause one object to move away from or towards another
orbit	cause one object to dodge around another
mulspeed	adjust velocity of an object with a multiplier
reflect	reflect an object away from another
setvelocity	give an object a velocity
squash	shorten an object in the direction of another
stretch	stretch an object in its direction of movement
stretchvel	cause an object to stretch as its velocity changes
stretchdist	stretch an object according to its distance from another
withdir/withvel	cause one object's direction/ velocity to be that of another

Space prohibits a detailed discussion of these various stimulus and response functions. However, to give a flavour we detail two of the response fuctions and show how they can be used in an animated sequence.

stretchvel *(ObjName, NormVel, wt)* makes the shape of an object dependent on its velocity

$$Stretch = 1 + wt(CurrentVel - NormVel)/NormVel$$

Where $NormVel$ is a parameter which determines the velocity for which the stretch factor will be 1.0, that is, it will not be stretched or compressed. The weighting factor, wt determines the degree of stretch. The $Stretch$ factor which is returned by this function is used to generate a degree of stretch in the direction of the object's motion.

stretchdist *(Obj1, Obj2, NormDist,wt)* makes the shape of an object dependent on its distance from another object.

$$Stretch = 1 + wt(CurrentDist - NormDist)/NormDist$$

Where $NormDist$ is a parameter which determines the distance for which the stretch factor will be 1.0.

Expressive Animation

The purpose of the ESCIM package is to enable us to test the hypothesis that, given the right primitives, it is easy to create expressive and animate motion. We present the following examples to illustrate successively how chasing, avoidance, and hitting can be achieved, and how they can be made expressive through the use of stretch and size control.

Chasing and Avoidance

The basic chasing algorithm is as follows (leaving aside the object definitions and color definitions). This animation script is presented in the ESCIM language.

```
/* if the chaser is getting close */
if (distance(chaser, avoider) < 0.3)
      /* The avoider accelerates.*/
      addspeed(avoider,0.03)
      /* The avoider turns away from the chaser.
      Direction is weighted 0.3 away from the chaser */
      away(avoider,chaser,0.3);
      /* Because this is a friendly chase the chaser slows.
      Newvelocity = oldvelocity times 0.7 */
      mulspeed(avoider,0.7);
endif;
/* if the chaser is far away */
if (distance(chaser,avoider) > 0.5) */
      /* the avoider can relax and slow down */
      mulspeed(chaser, 0.70);
      /* the chaser accelerates to catch the avoider. */
      addspeed(chaser, +0.02);
endif;
/* The chaser tends to move towards the avoider. */
towards(chaser, avoider, 0.4);
```

This is a basic chasing algorithm, it produces an endless catchup/avoid cycle. The point is that it looks like chasing, not like a following or pushing or pulling which have other algorithms. The chase algorithm above is sufficient, however, it is not very useful because it is not contained – the chase quickly escapes the boundary of the screen. In practice, some kind of wall avoidance is necessary such as

```
if (distance(avoider,walls) < 0.26)
      away(avoider,walls, 0.4)
      addspeed(avoider, +0.01)
endif
```

Because the chaser follows the avoider we do not have to worry about it getting lost.

While the animation described is convincing it has little pizazz. To make it more expressive we add the following

```
/* the chaser elongates when going fast
and squishes up short when going slow */
stretchvel(chaser, 0.05, 0.5);
/* the avoider elongates when going slowly
and squishes up short when going fast */
stretchvel(avoider, 0.14, -1.2);
```

The effect of these velocity dependent stretch functions is hard to describe verbally. But it is clear that the animation produced is more lively and "animate" and it is also clear that swapping the stretch functions for the two actors gives a result which is less like chasing and less expressive. The finding that shape change enhances the percept is in accordance with the observation of Michotte (1963) that having an object change shape as it moved makes it much more likely that that object will be perceived as animate. Figure 1 is a diagram which illustrates the motion cycle and the effect of stretch, although this in no way illustrates the perception of chasing.

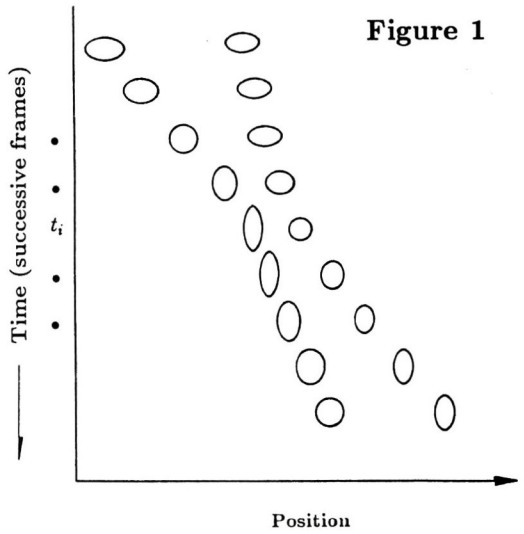

Figure 1

Hitting

There is more than one way to move an object from A to B. It can be chased, if it is animate; if it is inanimate, it can be hit like a volleyball.

The script for hitting has the following algorithm.

```
/* If the chaser would strike the ball before next frame */
if (timetoreach(chaser, ball) < 0.5)
      /* the chaser stops */
      mulspeed(chaser,0.01)
      /* and the ball takes off */
      addspeed(ball,0.1);
      /* in a direction away from the chaser */
      away(ball,chaser,1.0);
endif;
/* If the chaser is far away */
if (distance(chaser,ball) > 0.5) */
      /* the chaser accelerates for the next hit */
      addspeed(chaser, +0.02);
endif;
```

/* The chaser always moves towards the ball. */
towards(chaser, avoider, 0.6);
/* finally, because the atmosphere is viscous,
the ball always slows

mulspeed(ball, 0.93)

This example provides a perfectly good, if somewhat expressionless, example of one blob, clearly animate, chasing and striking another blob, which is clearly inanimate. To make the example more expressive we make the chaser compress as it gets closer to the ball.

stretchdist(chaser,ball, 0.08, 0.2)

This makes the chaser look as though it is compressed by the impact. The hitting example is illustrated graphically in figure 2.

An important point here is that there is no attempt here to physically model the collision of one body with another, as in the work of Terzopoulos and Witkin (1988). We are trying to show that an appropriate percept can be generated with simple functions, at a low cost and without any understanding of physics or the behaviour of volleyball players.

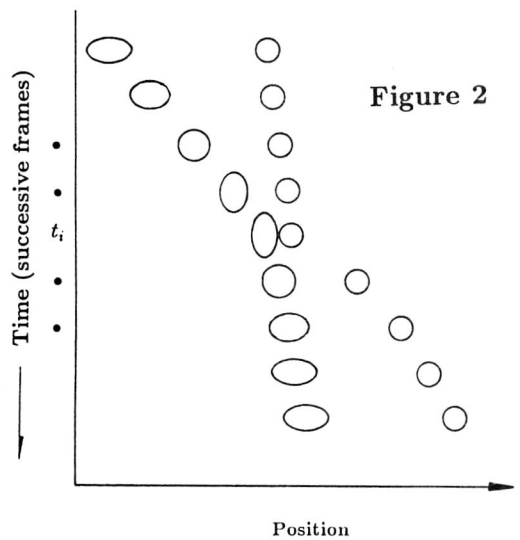

Figure 2

Discussion

With stimulus-response animation and the appropriate primitives it is possible to generate simple rules and watch fairly elaborate and compelling social interactions evolve before one's eyes. In some simple programmed environments, changes in behavioural rules may change social interactions, sometimes in dramatic ways, other behaviours are robust and survive large parameter changes. Playing with these systems is fun and we advocate it for its own sake.

There are also obvious applications in computer animation for the control of the behaviour of background actors, such as flies, or birds Reynolds (1982, 1987). It may not be necessary to have direct control over such actors, but they can be made to respond to other actors whose motion is controlled using other techniques.

Another possibility is to use motion metaphors to convey information about the state of a system. Disney style character animations convey a vaste amount of visual information and it seems likely that some of the techniques might be adaptable to data display. This area remains largely unexplored by interface designers, but it is clear that the way in which an object moves can convey important information such as liveness causality, and intentionality. All of these are concepts which may be hard to express using static techniques. Often, when objects are shifted from one place to another on a screen it is useful to have them move from A to B, rather than disappear at A and reappear at B, which can be confusing. Motion gives the perception of object continuity. However, as we have seen there are many ways of moving an object other than linear interpolation over time. An object can be pushed pulled, hit, prodded or chased into position, and all of these modes are perceptually distinct and can be used to convey information about dominance, continuity object constancy and control. Our plan is to use the ESCIM system as a vehicle for studying the use of motion metaphors using S-R animation techniques.

References

Basili J.N., 1976, Temporal and spatial contingencies in the perception of social events, Journal of Personality and Social Psychology, 33, 680-685.

Braitenberg, 1984, Vehicles: Experiments in Synthetic Psychology, The MIT Press, 1984.

Lethbridge, T.C. and Ware, C. 1987, Animation Using Behavior Functions, Proceedings of 1987 Workshop on Visual Languages Linkoping, Sweden, Also to appear in S.K. Chang, Ed. Visual Languages, Plenum Press.

Lethbridge, T.C. and Ware, C. (in press), A Simple Heuristically Based Method for Expressive Stimulus Response Animation. Computers and Graphics.

Marion, A., Fleisher, K. and Vickers, M. 1984, Towards Expressive Animation For Interactive Characters, Proc. Graphics Interface 84, 17-20.

Michotte, A. 1963, The Perception of Causality. Methuen, 1963.

Reynolds, C.W. 1982, Computer Animation with Scripts and Actors, Computer Graphics, 16, 289-296.

Reynolds, C.W. 1987, Flocks, Herds, and Schools: A Distributed Behavioural Model, Computer Graphics, 21, 25-34.

Terzopoulis, D, and Witkin, A., 1988, Physically-Based Models with Rigid and Deformable Components. Proceedings of Graphics Interface '88, 146-154.

Wilhelms, J. 1987, Towards Automatic Motion Control. IEEE Computer Graphics and Applications, 7, 11-22.