A General Specification for Scene Animation

GENE S. LEE

Department of Computer Science
University of British Columbia
Vancouver, BC Canada V6T 1Z4
gslee@cs.ubc.ca | gslee@alum.mit.edu

Abstract. This paper presents the foundations of IBAL, a general specification for scene animation. IBAL regulates interactions among graphical elements, such as geometric models and behavioral algorithms, to describe animated scenes. Interactions among graphical elements are managed by primitives which are arranged into three graphs. The three graphs logically partition an animation into "who," "what," and "when." IBAL's unique design permits applications to represent, save, and restore a wide variety of complex scenes. A short summary of existing formats for scene modeling and scene animation is presented to distinguish and to assess the utility of IBAL's design.

Keywords: Scene Animation, General Specification, Software Interaction

1 Introduction

Computer animation is the development and production of computer animated scenes. Animated scenes consist of many graphical elements interacting over time. Graphical elements are geometric models, algorithms, and data structures that define the state of an animation and control its gradual evolution from one state to the next.

Many tools exist to display, store, and edit quickly various graphical elements. File formats are one type of tool that are essential and very popular. File formats adhere to specifications to store information onto data files. They establish a means for applications to store and to share the graphical elements of a scene. Applications that use established specifications are bound to read and write information according to specific guidelines, but they are free to represent and manipulate the information using their own means and data structures.

Many types of specifications for computer graphics exist. However, only specifications that store and represent geometric models, such as 3DMF [3], IGES [13], and DXF [4], experience widespread success. Specifications for animation are many, but few are popular or standardized. Those that are popular, such as Quicktime [12] and MPEG [9], simply store the resultant frames of animations. They neither describe nor control the contents of the animations. From the images alone, it is arduous to obtain or identify all the graphical elements that define the animations.

Of the many specifications for animation that do describe the contents and behaviors of animated scenes, few are popular. For most specifications, such as VRML [5], rapid development, not reuse or portability, is the primary goal. Thus, the specifications are often difficult to partition into logical parts and arduous to use as scene complexity grows. It is not always easy to identify, extract, or replace the parts of animation that define "who," "what," and "when." [20] Ideally, a robust specification manages complexity, accomodates numerous means to animate, and localizes control structures. These key features are essential for widespread acceptance and easy use.

It is the aim of this paper to present the foundations of IBAL, an Interaction-Based Animation Language. IBAL is a general, implementation-free specification for scene animation. It relies upon first-class primitives to organize and control the interactions of graphical elements. As graphical elements interact, they exchange information and update state. The specification consists of three graphs. The first graph describes scene elements; the second graph describes dynamic behaviors; and the third graph describes temporal properties. The three graphs readily separate animations into logical parts and reduce complex scenes to a hierarchy of interactions. With IBAL, a wide variety of scenes and animation techniques can be shared and reused.

This paper consists of six sections. Section two presents an overview of various specifications for computer animation. It briefly discusses several issues that determine the success of popular specifications for modeling and uses these issues to assess the quality of existing specifications for animation. Section three presents an overview of IBAL and describes the basic design of its three graphs. Section four demonstrates IBAL's utility by providing an example and comparing it with other specifications. Sections five and six analyze IBAL's design and summarize the results of this work.

[0] *Presented at Sibgrapi '98, International Symposium on Computer Graphics, Image Processing, and Vision, Oct 1998, Rio De Janeiro, Brazil*

2 Background

The graphics industry abounds with many proposed specifications for graphics and animation. It is common for applications to support many specifications for scene modeling and only one or two specifications for scene animation. It is relatively easy for applications to share descriptions of models, but not descriptions of animated scenes. Many of the issues that influence the success of modeling specifications are not satisfied by existing specifications for animation. This section describes these issues and uses them as a basis for analysis of and comparison with various types of specification for scene animation.

2.1 Specifications for Scene Modeling

Specifications for scene modeling, such as Inventor [26] and OBJ [1], describe the shape and physical characteristics of static scenes. They are popular and standardized for five reasons:

- *There is a great need to share information among many applications and to form archives of scene information.*

Sharing and archiving scene information is important. Not all applications provide the same features and interfaces to construct and interact with static scenes. Applications that share reap the benefits offered by others. Archives emphasize scene integration over scene development. Scenes are produced by browsing libraries of data, such as those provided by Viewpoint Datalabs, Inc. [27] and New World Graphics, Inc. [23], and by adapting acceptable elements.

- *There is a commonly accepted set of low-level (and some high-level) data structures for scene building.*

Without regard for efficiency, it is possible to represent all scenes with basic structures, such as points, lines, polygons, and splines, or with simple polyhedra, such as cubes, spheres, and cylinders.[1] Often, the design benefits of alternative structures are lost when scenes are transformed into basic structures. However, this drawback is universally accepted because it permits all applications to produce similar reproductions of original scenes.

- *There are acceptable ways to form methodologically static scenes from logical parts (with the basic data structures).*

Many specifications commonly separate scenes into many logical parts, such as sets of points, normals, faces, and colors. Scenes built from parts are easy to rebuild, integrate, and dissect. Applications which load scenes can rapidly select and discard unwanted parts.

[1] Representing high precision scenes with basic structures often involves some reduction in quality. For many users, approximate representations are acceptable and sometimes preferred.

- *There are ways to build and to manage complex scenes effectively.*

Many specifications employ logical operators or hierarchical schemes to compose complex scenes from simpler models. Composed scenes are easier to update and frequently easier to store.

- *It is possible to model a wide variety of static scenes.*

With basic structures and logical operators, it is possible to describe the shape and characteristics of all graphical elements within a scene. Not all scene descriptions may be efficiently described, but they are representable.

2.2 Specifications for Scene Animation

2.2.1 Taxonomy

As described elsewhere [18], specifications for scene animation consist of *appearance* formats and *content* format. Appearance formats store frames (images) of scenes while content formats store descriptions of scenes. Each major type further decomposes into several subtypes. Collectively, all these types form a class hierarchy, as shown in Figure 1.

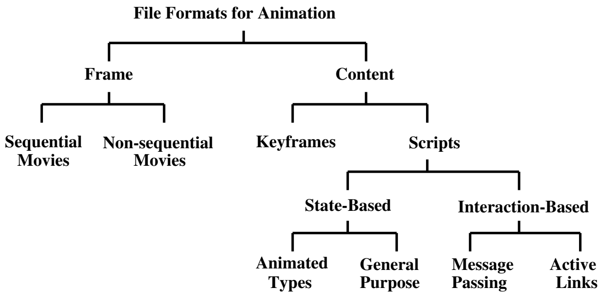

Figure 1: A Taxonomy of Specifications for Scene Animation

Appearance formats divide into two related subtypes, *sequential movies* and *non-sequential movies*. The former stores a linear progression of images while the latter, a collage of images. Neither type is suitable for storing or obtaining information about the contents of an animated scene. Content formats also divide into two subtypes: *keyframes* and *scripts*. Keyframes specifications [2] store sequences of key values that explicitly establish state information while scripts store sequences of expressions that compute state information.

Scripts further divide into *state-based* and *interaction-based* formats. State-based formats, consisting of *animated types* [16] and *general purpose* [22, 6] formats, explicitly manipulate the states of scene elements. Expressions intermix the parameters of scene elements with special operators to manage the evolution of state values. Interaction-based formats [5], consisting of *message-passing* and *active links* formats, identify relationships between scene elements. Related scene elements change state as they exchange information.

2.2.2 Analysis of Types

The chart in Table 1 compares the three major categories of formats that store content. The comparison indicates that none of the categories completely address the issues that popularize modeling formats.

Keyframe formats are useful and easy to use because they establish a logical partitioning of information: keyframes update state variables. Unfortunately, keyframes are arduous to create when specifying scenes of great complexity. Few approaches exist to build higher level abstractions upon the keyframes and no simple means exist to create animations that adapt to run-time. The state of a keyframe animation is always set before it begins.

State-based script files are useful to describe a wide variety of animations. They can carefully guide animations, like keyframe files, and they can form relations between graphical elements to establish constraints and adaptive behaviors. However, they are not easy to separate into parts. Most employ concise representations to describe and to control state information. It is not always easy to extract and to replace quickly only the expressions from a script that modify (or compute) state. Most often, these expressions are tightly intertwined with other expressions that establish state information and specify temporal properties.

Like keyframe files, interaction-based script files partition an animation into two basic parts: those that represent state and those that modify state. Software interactions, such as function calls or method invocations, moderate the flow of information between the two parts. Both file types are easy to decipher and separate into their parts. However, interaction-based scripts model a greater variety of scenes. They provide a greater variety of abstractions and a greater number of part mappings. Unfortunately, they lack rules and patterns for program development (and reuse). Few structures exist to organize many software interactions and to control the evolution of interactions over time. Users must employ their own means or other scripting languages to manage the interactions. Thus, many interaction-based scripts, such as VRML [5], encounter problems similar to those of state-based scripts. As scene complexity grows, it becomes arduous to partition animations into parts.

3 A General Specification for Animation

3.1 Overview

IBAL is an interaction-based format for scene animation. It establishes a general specification to describe the contents and behaviors of time-varying scenes. The specification bases its design upon the IMVCD (Informer - Model - Viewer - Controller - Delegator) modeling framework of the RASP toolkit [20]. IMVCD partitions scenes into primitives for scene modeling and primitives for scene animation. The latter primitive set animates scenes by forming interactions between the modeling primitives. IMVCD identifies the basic elements of an animation which IBAL specifies and controls.

IBAL partitions an animation into three animation graphs: a *scene graph*, an *action graph*, and a *time graph*. The three graphs identify separately the “who,” “what,” and “when” of an animation. The scene graph identifies the graphical elements of an animation. It describes scene elements and their properties, such as shape, surface characteristics, and viewing parameters. The action graph defines the interactions among graphical elements. It organizes collections of interactions to form behaviors, actions that produce state changes. The time graph defines a temporal ordering of behaviors. It indicates how behaviors relate and execute over time.

3.2 Framework

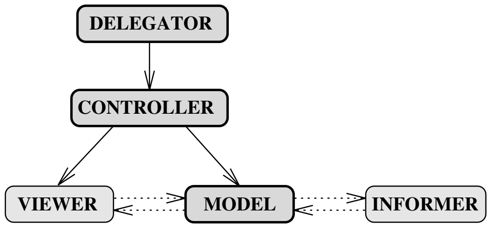

Figure 2: The IMVCD modeling framework

The IMVCD modeling framework abstractly decomposes computer animations (and other visual simulations) into five key components: *informers*, *models*, *viewers*, *controllers*, and *delegators*. As seen in Figure 2, the informers define the properties of models which the viewers visualize and the controllers control. The delegators decide when controllers act and which models and viewers the controllers command. Like popular frameworks, such as MVC [17] and Interviews [21], IMVCD focuses on the architecture of programs, as opposed to the reuse of implementation. It emphasizes the communicational paths between components, not the dataflow between them. IMVCD does not define abstract components for other components to derive functions and properties. IMVCD is an implementation-free framework that identifies, not evaluates, the relationships between software components.

IBAL’s components match those described by IMVCD. Models and informers, organized by IBAL’s scene graph, identify graphical elements. Controllers, identified by IBAL’s action graph, manage interactions that produce state changes. Delegators, formed with IBAL’s time graph, vary the behavior of interactions over time. With

	CONTENT FORMATS FOR SCENE ANIMATION			
	Keyframes	**State-based scripts**	**Interaction-based scripts**	**IBAL scripts**
Need to Share	Yes	Yes	Yes	Yes
Foundation	Keyframe	Expression	Expression	Expression
Separable Parts	Yes	No	Yes	Yes
Effective Development	Low	Some	Low	Yes
Scene Complexity	Limited	Unlimited	Unlimited	Unlimited?

Table 1: A comparison of specifications for scene animation

IMVCD, IBAL's specification is easy to understand and easy to augment. New elements that are consistent with IMVCD freely replace or complement existing elements in any IBAL script.

3.3 Scene Graph

IBAL's scene graph resembles the scene graph of VRML. As illustrated in Figure 3, it consists of a hierarchy of nodes that represent geometries, transformations, surface properties, and viewing parameters. The hierarchical arrangement of nodes forms relations for a (rendering) process to interpret.

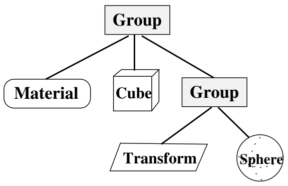

Figure 3: Two conflicting scene graphs

Unlike VRML, IBAL's scene graphs are not immediately realized after they are parsed. IBAL's scene graphs are hierarchical representations that take form during runtime. At run-time, the action and time graphs determine which scene graphs are active. Thus, unlike VRML, IBAL files may contain multiple scene graphs with identical elements that appear to conflict. Two conflicting graphs, as shown in Figure 3, are difficult to maintain in VRML without employing many "switch" nodes. For large numbers of arrangements, switch nodes are troublesome and difficult to manage dynamically.

3.3.1 Ports

In IBAL, all scene graph nodes possess ports. Ports are first-class structures that moderate the flow of information between a node and its environment. First-class interactions, established in IBAL's action graph, establish links between the ports of nodes to foster communications. Communications occur when the interactions receive messages from IBAL's time graphs. Upon receiving notification to interact, the ports either accept data to update the node's state or disseminate data to identify the node's state.

The ports of IBAL and the fields of VRML are functionally equivalent, but they are not the same. Although both are first-class and bond with elements that cause interactions, ports are high-level abstractions. Ports are analogous to the member functions of object classes while fields are analogous to data members of object classes. Ports hide the internal members of a node and perform operations, while fields are publicly accessible data members that simply send or receive state information. Ports, unlike fields, employ operations to compute state. In addition, multiple ports may modify the same internal state. Each port can provide a differing service. Multiple fields represent multiple states. It is neither possible nor acceptable to devise two fields that reference the same state variable.

3.4 Action Graph

IBAL's action graph describes the activities of an animation. Activities are collections of interactions that produce a particular behavior. Behaviors may express simple actions, such as object translations, or complex actions, such as ambulatory motions. Activities operate over an interval of time. They act continuously from beginning to end. IBAL's time graph determines when and under what conditions the activities operate.

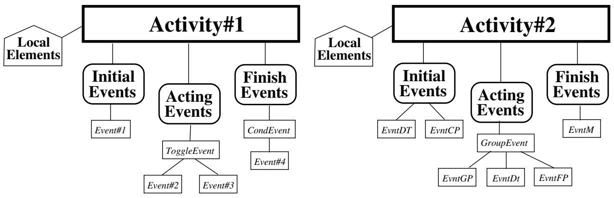

Figure 4: Action Graph

Every activity, as illustrated in Figure 4, consists of four parts. Part one specifies all node elements that are local to the activity. Local elements are either *controllers* or *virtual ports*. Controllers are functional units that perform (high-level) operations. With the assistance of interactions, they animate the states of models. Controllers operate on internal variables which interactions copy and disseminate. Interactions link controllers to models (and other controllers) by their ports. Information flows among the ports as the interactions operate. Virtual ports are free-floating ports that form symbolic relations between real ports — those associated with models or controllers. They establish one-way constraints that are evaluated when interactions require information from them.

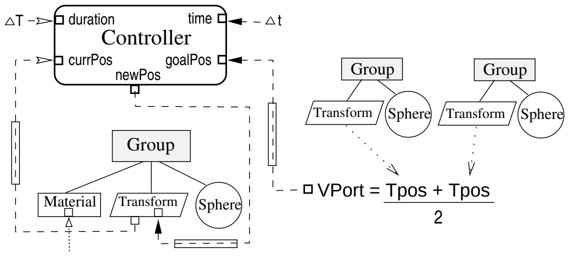

Figure 5: The interactions of a scene that moves one ball towards the center of two others.

The diagram in Figure 5 illustrates the interactions of a scene involving a controller, a virtual port, and three models. The controller moves one of the models towards the center of the other two. Interactions manage the flow of information between the ports of the models and the controller. The virtual port combines real ports with mathematical operators to compute the center of the two models. Each time an interaction accesses the virtual port, the center of the two models is computed.

Parts two, three, and four of an activity consist of a hierarchy of interactions, referred to as *events*. Nodes of the hierarchy manage either interactions between ports or group-related interactions. Event nodes verify that communicating ports are compatible and available. Group nodes link interaction by selection, association, or aggregation. Each of the three parts organize a separate hierarchy with an alternate activation time. Part two's hierarchy, referred to as *initial events*, operates when the activity begins. Part three's hierarchy, referred to as *acting events*, triggers continuously for the activity's entire duration. Part four's hierarchy, referred to as *finish events*, operates immediately after the activity terminates.

Activity#2 of Figure 4 shows the three event sets of the action shown in Figure 5. The initial events, drawn with clear arrowheads, provide the controller with a temporal duration and the current position of the primary model. The temporal duration, established in a time graph, informs the controller how much time it is given to operate. The acting events, drawn with dark arrowheads, perform two operations with three events. The first two events provide the controller with a time step and a location defined by the virtual port. The time step notifies the controller of the passing of time. As the activity acts, the time step starts at zero and gradually evolves to the length of the activity's duration. The third event passes information from the controller to the primary model. Each time the event acts, it updates the primary object's position. The one finishing event, drawn with a short, clear arrowhead, updates the primary model's material characteristics when the activity terminates.

Kind	Type
Simple	Event; StateEvent; CallEvent
Time	TimeEvent; StepEvent; DurationEvent; TimeStepEvent;
Logical	BoolEvent; CondEvent; BiCondEvent;
Group	ToggleEvent; ChainEvent; SwitchEvent;
Scene	SceneOnEvent; SceneOffEvent;

Table 2: Kinds of event types

Kind	Type
Timer	Interval; ScaleInterval; ShiftInterval; Granularity; ScaleGranularity
Relation	Stop; Start; Meet; Delimit; Sequence
Modifier	Condition; Reactivate; Inactive

Table 3: Kinds of time graph nodes

3.4.1 Events

As shown in Table 2, IBAL currently specifies five kinds of events types: *simple*, *time*, *logical*, *group*, and *scene*. Simple events apply basic techniques to manage ports. They either monitor them, pass data between them, or instruct them to invoke commands. Time events disseminate temporal information. They provide ports with timing values, such as current time, expected duration, and future time step. The states of components advance as they receive timing values. Logical events relate ports to logical flags and conditions. The state of the logical conditions determines how time graph primitives interpret the events. Group events contain other events. They selectively toggle, chain, or order events to bind them closely. Scene events toggle the states of scene graphs. They turn scene graphs on and off.

3.5 Time Graph

IBAL's time graph describes the temporal characteristics of an animation. As shown in Figure 6, it consists of a hierarchy of nodes that establish a *timing pattern* for concurrent activities. The timing pattern indicates how activities relate and when activities occur. The top node of every hierarchy establishes an initial reference interval. The interval defines a span of time upon which the nodes of the hierarchy act. Nodes either modify the interval or derive new intervals. Activities within the hierarchy receive as their duration the interval which their parent assigns.

As shown the Table 3, a time graph consists of three kinds of nodes: *timers*, *relations*, and *modifiers*. Timer nodes identify timing values. They update the span or granularity of the current interval of time. An interval's granularity determines the frequency at which updates to activities occur. Activities with high granular values update with greater frequency. Relation nodes form temporal relations among groups of activities. Temporal relations, such as simultaneous start and stop, establish time-based relationships. As activities operate, they influence the operation of other activities. For example, two activities "meet" if the termination of the first causes the activation of the second. Modifiers assign conditions to relation nodes. The conditions relate the operation of the nodes to certain states. The nodes operate only when the specified states arise. For example, the modifier "inactive" indicates that relations nodes affect only inactive activities. For active activities, the node remains dormant.

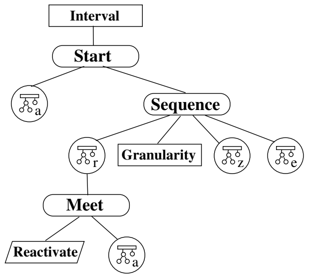

Figure 6: Time graph

An example of a time graph with all three nodes types appears in Figure 6. The `interval` node establishes an interval of time for the `start` node. `Start` informs activity `a` and the `sequence` node to begin simultaneously. The `sequence` node sequences through activities `r`, `z`, and `e`. The `granularity` nodes modifies the update frequencies of activities `z` and `e`. The `meet` node, with the modifier `reactivate`, indicates that the activity `a` restarts when activity `r` finishes.

4 Applications

Numerous examples highlighting the use of IBAL have been devised with the RASP toolkit for animation [20]. As seen on the RASP webpage[2], one of these examples, called "Catch," presents an animation in which two robotic arms continuously play catch with a ball. Catch consists of three models, one for each robot and the ball, and eight activities, two for the ball and three for each arm.

The scene graph of Catch is a simple hierarchy of shapes, transforms, and material properties. The transforms modify the positions and orientations of the ball and the two arms.

The action graph of Catch consists of separate activities for the ball's position and direction, and separate activities for the arm's position and stance. An additional

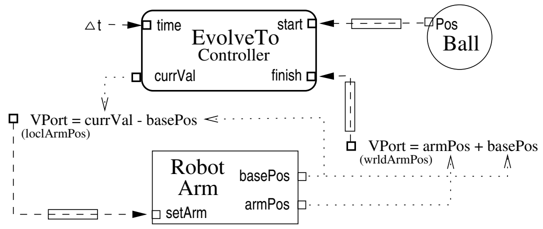

Figure 7: Activity `mvArmR`

activity exists for each arm to indicate whether or not the arm is in contact with the ball. Figure 7 illustrates the interactions of the activity `mvArmR` that adjusts the stance of an arm. The activity consists of four acting events and three local elements, one controller and two virtual ports. The controller produces a sequence of values from start to finish. Each time it receives a time step, the controller produces a new value. The virtual ports transform the arm's position from local space to world space, and vice versa. Each time the activity updates, the four events pass information among the ball, the robot, and the controller.

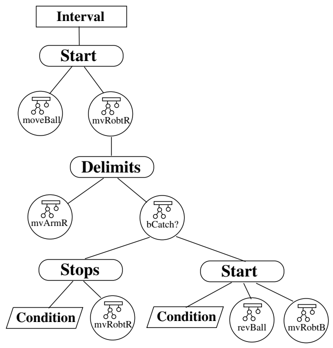

Figure 8: Time graph of Catch

The time graph of one robotic arm from Catch is shown in Figure 8. The time graph simultaneously starts both the movements of the ball and the arm. The activity that governs the arm's position delimits `mvArmR`, the activity that governs the arm's stance, and `bCatch?`, the activity that determines if the arm is in contact with the ball. If the arm and the ball meet, the arm halts, the ball changes direction, and the other robot starts. Once the activity `mvRobtR` stops, the two activities that it delimits also stop.

IBAL's design logically partitions Catch into intelligible, modular parts. To replace or augment the animation, elements are either introduced or extracted from any of the three graphs. Existing specifications, such as keyframing, state-based scripts, and VRML, do not offer such a convenient breakdown. They opt to integrate tightly all expressions to produce concise representations. The representations are often smaller, but usually less adaptable and not as easy to share. For example, the code

[2] www.cs.ubc.ca/spider/gslee/Rasp/rasp.html

```
while(1) {
    if (red == TRUE) {
            robot = redArm;
            red = !red;
    }
    else
            robot = blueArm;
    while(1) {
            move ball & robot's base and arm
            compute distance between arm and ball
            if (distance < 0) {
                  reverse ball's direction;
                  break;
            }
    }
```

Figure 9: Description of Catch with a fictional state-based script.

segment in Figure 9 presents a description of Catch with a fictional general-purpose, state-based script. The description is smaller, but not as useful. The organization of the expressions implicitly defines many properties that IBAL explicitly identifies. It is difficult to simply modify just the controllers or the temporal attributes of the animation.

5 Analysis

5.1 Five Issues

The IBAL specification attempts to satisfy the five issues that influence the success of specifications for scene modeling. It consists of a basic set of low-level data structures for scene modeling, activity development, and temporal coordination. The partitioning of the data structures is logical and intelligible. IBAL's three graphs separately identify the three parts of all animated scenes: who, what, and when. The hierarchical organization of the three graphs effectively builds and manages the growth of complex animations. Each level of the three hierarchies refines or augments its preceding level. Preliminary results [19, 20] indicate that IBAL can model a wide variety of animated scenes. Many animation techniques, such as keyframing, kinematics, and behavioral modeling are relatively easy to specify and control. However, greater testing is necessary to determine if IBAL's design is acceptable for widespread use.

5.2 Related Work

IBAL's design shares commonalities with many previous works in graphics. Interaction-based tools, such as CONMAN [11] and VTK [25], interconnect components to compose formulas and algorithms, while other graphics tools, such as CONDOR [15], T-BAG [7], and VB2 [10], interconnect components to form constraints. However, unlike IBAL, these other tool sets provide few primitives to control interactions dynamically or to manage interactions hierarchically. The interactions are neither first-class nor separate from the components they interconnect; thus, they are difficult to identify and to extract for reuse. In addition, most tools lack branching structures that can form conditional interactions. Most interactions are devised to operate unconditionally.

UGA [29] and ASAS [24] manage interactions by passing messages among components. Interactions transpire immediately after messages pass or when special times arise, indicated by "timestamps." Unlike IBAL's approach, messages passing schemes do not readily highlight the flow of information among parts. The messages, which appear throughout the body of a script, do not easily identify the relationships that exist among numerous interactions. The relationships are often dispersed and occasionally difficult to discern.

Previous works by Fiume [8] and Kalra [14] foster a coordination scheme that closely resembles IBAL's time graph. Both works employ temporal operators to produce a temporal ordering of events and behaviors. However, unlike IBAL, neither work advocates a simple graph-based method to order behaviors nor proposes an approach to devise the behaviors that it orders. In addition, Karla's work lacks methods to apply conditions to operators and Fiume's work lacks operators to assign explicit temporal values to behaviors.

5.3 Limitations

The IBAL specification is not without its limits. It does not currently specify two important constructs for animation (and simulation). First, it does not specify any means to update dynamically any of its three graphs at run-time. Once the three graphs are traversed, they are fixed. Run-time modifications are useful to model very complex scenes that adapt to run-time conditions. It is unacceptable to expect all implementations of IBAL to devise their own useful mechanisms to update any of the three graphs. Augmenting IBAL with non-standard mechanisms defeats its primary purpose, sharing. Sharing is difficult if each implementation derives it own methodology to update the graphs.

Second, IBAL does not specify any means to spawn *run-time actions*. Run-time actions occur in response to run-time conditions. It is impossible to infer exactly how many of these actions arise and at what times they occur. Run-time actions are essential for blending simulation technologies with animation techniques. In process-oriented simulations [28], actions spawn as special events, referred to as transactions, arrive. Each arrival initiates a sequence of steps that usually place the transaction in a queue. In animation, explosions are good examples of run-time actions. They are unpredictable and may occur often. To handle each explosion separately is tedious and inefficient. It is better to specify an action that spawns each time an explosion occurs.

6 Conclusion

This paper has presented the foundations of IBAL, a general specification for scene animation. IBAL relies upon first-class primitives to manage the interaction among graphical elements, such as the models and algorithms of an animation that possess or update state information. As the graphical elements interact over time, they cause scenes to animate. IBAL partitions animations into three graphs. The graphs indicate who interacts, what interactions form, and when interactions occur.

A brief analysis of the existing specifications for scene modeling and scene animation is presented to distinguish IBAL's unique design. IBAL addresses many of the issues that popularize specifications for scene modeling. It provides a basic set of low-level data structures for animation that effectively describes a wide variety of complex scenes.

Preliminary results indicate that IBAL's design is useful. Numerous animations, such as the one presented in this paper, exemplify IBAL's utility. However, more testing and development is required to assess IBAL's overall value. Future plans are underway to augment IBAL with methods to update dynamically the configuration of scene elements and to spawn actions at run-time. These methods will augment IBAL's ability to integrate simulation techniques with computer animation.

Acknowledgments

The author would like to thank Marcelo Walter and Dave Forsey, for their helpful comments on content, and Gia Lee, for her helpful comments on grammar and style.

References

[1] ALIAS!WAVEFRONT, INC. *Advanced Visualizer Object Format*, 1997.

[2] ALIAS!WAVEFRONT, INC. *Scene Description Language (SDL) Reference Manual*, 1997.

[3] APPLE COMPUTER INC. *3D Graphics Programming With QuickDraw 3D*. Addison-Wesley, Reading, MA, 1995.

[4] AUTODESK, I. *Drawing Exchange Format (DXF)*, autocad release 1.0 ed., 1982.

[5] BELL, G., CAREY, R., AND MARRIN, C. *VRML 2.0 Specification*, version 2.0 ed., 1996. ISO/IEC CD 14772.

[6] ELLIOTT, C., AND HUDAK, P. Functional Reactive Animation. In *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming* (Amsterdam, The Netherlands, June 1997), pp. 263–273.

[7] ELLIOTT, C., SCHECHTER, G., YEUNG, R., AND ABI-EZZI, S. TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications. In *Computer Graphics* (Aug 1994), ACM SIGGRAPH, pp. 421–434.

[8] FIUME, E., TSICHRITZIS, D., AND DAMI, L. A Temporal Scripting Language for Object-Oriented Animation. In *Eurographics '87* (North-Holland, 1987), Eurographics Association, Elsevier Science Publishers B. V., pp. 283–294.

[9] GALL, D. L. MPEG: A Video Compress Standard for Multimedia Applications. *Communications of the ACM 34*, 4 (1991), 46–58.

[10] GOBBETTI, E., AND BALAGUER, J. An Integrated Environment to Visually Construct 3D Animations. In *Computer Graphics* (Aug. 1995), ACM SIGGRAPH, pp. 395–398.

[11] HAEBERLI, P. ConMan: A Visual Programming Language for Interactive Graphics. In *Computer Graphics* (1988), ACM SIGGRAPH, pp. 103–111.

[12] HONE, R. *QuickTime*, 2nd ed. Prima Publishing, Sept 1994.

[13] IGES/PDES ORGANIZATION. *Initial Graphic Exchange Specification (IGES)*, version 5.3 ed., 1996. ANS US PRO/IPO-100-1996.

[14] KALRA, D., AND BARR, A. H. Modeling with Time and Events in Computer Animation. In *Eurographics '92* (Cambridge, UK, Aug 1992), Eurographics Assoc., pp. 45–58.

[15] KASS, M. CONDOR: Constraint-Based Dataflow. In *Computer Graphics* (Chicago, IL, July 1992), ACM SIGGRAPH, pp. 321–330.

[16] KOVED, L., AND WOOTEN, W. L. GROOP: An Object-Oriented Toolkit for Animated 3D Graphics. In *OOPSLA '93* (1993), ACM, pp. 309–325.

[17] KRASNER, G. E., AND POPE, S. T. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming 1*, 3 (Aug/Sept 1988), 26–49.

[18] LEE, G. S. A Classification of File Formats for Animation. In *Western Symposium on Interactive Graphics* (Whistler, Canada, 1998).

[19] LEE, G. S. Reusable Interactions for Animation. In *5th International Conference of Software Reuse* (Victoria, Canada, 1998), IEEE Computer.

[20] LEE, G. S., FORSEY, D., AND PAI, D. RASP - A Visual Simulation Platform. *Submitted for publication to IEEE Transactions on Visualization and Computer Graphics* (1997).

[21] LINTON, M. A., VLISSIDES, J. M., AND CALDER, P. R. Composing User Interfaces with InterViews. *Computer 22*, 2 (Feb 1989), 8–22.

[22] NAJORK, M. A., AND BROWN, M. H. Obliq-3D: A High-Level, Fast-Turnaround 3D Animation System. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June 1995), 175–193.

[23] New World Graphics, Inc. URL: www.nwginc.com.

[24] REYNOLDS, C. Computer Animation with Scripts and Actors. In *Computer Graphics* (July 1982), ACM SIGGRAPH, pp. 289–296.

[25] SCHROEDER, W. J., MARTIN, K. M., AND LORENSEN, W. E. The Design and Implementation of an Object-Oriented Toolkit for 3D Graphics and Visualization. In *Visualization '96* (Oct. 1996), IEEE.

[26] STRAUSS, P. S., AND CAREY, R. An Object-Oriented 3D Graphics Toolkit. In *Computer Graphics* (Chicago, IL, July 1992), ACM SIGGRAPH, pp. 341–349.

[27] Viewpoint Datalabs, Inc. URL: www.viewpoint.com.

[28] ZEIGLER, B. P. *Theory of Modelling and Simulation*. John Wiley & Sons, New York, 1976.

[29] ZELEZNIK, R. C., CONNER, D. B., WLOKA, M. M., ALIAGA, D. G., HUANG, N. T., HUBBARD, P. M., KNEP, B., KAUFMAN, H., HUGHES, J. F., AND VAN DAM, A. An Object-Oriented Framework for the Integration of Interactive Animation Techniques. In *Computer Graphics* (1991), ACM SIGGRAPH, pp. 105–111.