Towards an Integration of Computer Simulation with Computer Graphics

Gene S. Lee
Department of Computer Science
University of British Columbia
Vancouver, Canada V6T 1Z4
gslee@cs.ubc.ca

Abstract

This paper presents an initial design to integrate the methods of computer simulation with the methods of computer graphics. It begins with a discussion on the strengths and weaknesses of the two fields, and progresses into a description of three key issues – *software architecture*, *software coordination*, and *indeterminant maps* – that hamper their integration. The paper present a possible solution to the first two issues and concludes with some comments of the third.

Keywords: Software Architecture, Software Coordination

1 Introduction

Research in computer simulation and computer graphics overlap greatly in many areas. Each employs numerical modeling methods to simulate the behavior of real (or non-real) world entities. Over the years, the simulation community has developed many methods to update these entities over time while the computer graphics community has produced many methods to visualize these entities as they change. An integration of these two methods presents many benefits, two of which are as follows:.

- It permits the computer graphics community to employ simulation techniques for animation. Common structures in simulation, such as petri nets and queues, introduce high level modeling methods to graphics that have not generally been provided before. Similarly, it permits the computer simulation community to employ graphics techniques for visualization. Common graphics methods, such as keyframing and texture mapping, produce visuals of greater depth and appeal than those produced simply from the presentation of static icons, as is commonly performed by some common tools for simulation, such as SIMGRAPHICS [21] and PROOF ANIMATION [3].
- It supports a foundation upon which to build a wide variety of *visual simulations*, simulations that visualize the behavior or execution of its simulated entities.[1] With this foundation, the computer graphics community would benefit from years of research in simulation, and, likewise, the simulation community would benefit from years of research in computer graphics.

This paper present the beginnings of an informal approach to integrate the methods of computer simulation with the methods of computer graphics. It begins with a short comparison of the two fields and identifies their strengths and weaknesses. Afterwards, it briefly outlines a plan to integrate the two fields. The plan identifies three issues – *software architecture*, *software coordination*, and *indeterminant maps* – that hamper their integration, and then, present a possible solution to the first two. The paper concludes with some additional comments and discusses future avenues of research.

2 Comparison

This section presents a brief comparison of computer graphics and computer simulation. The first subsection discusses traditional differences while the second subsection identifies what elements each of field lacks in preparing a solid foundation for visual simulation.

2.1 How do they differ?

The computer graphics community produces computer animations – plausible, visual simulations. The simulations emphasize aesthetics, interaction, and control, over computation and analysis. For example, *key-framing*, a popular control technique, interpolates key values to compute intermediary values. The outcomes of the simulations are known before they begin. The aim of visualization is to present the results of controlled computation. As computations compute, visuals animate. In contrast, each time step in a visual simulation updates the values that effect computation. A free-falling object's position updates as time progresses, and an articulated figure grabs as an planning algorithm calculates. For many visual simulations, there exists a simple mapping between variables and visual attributes. As the variables update, the visual attributes change.

The computer simulation community creates visual simulations to verify, present, and validate mathematical models. They stress computation, not control or aesthetics. Simulations establish initial conditions, execute state changes, and monitor results. Unlike their counterparts in computer graphics, they observe, not control, the production of results. Simulations execute repeatedly to ensure they observe the proper relationship between initial conditions and final results. Attempts to integrate general simulation techniques with computer graphics have produced mixed results. Many attempts, such as those based loosely on physics [25], serve limited use. It is difficult to predict and to vary the outcome of such simulations. However, some integrated approaches permit great flexibility. Simulations, such as those based on L-systems [18] or particle systems [19], provide adequate controls for users to carefully produce deterministic results.

2.2 Strengths and Weaknesses

2.2.1 Computer Graphics

Many of the early methods in computer graphics emphasized the development and production of static, high quality visual images. From these methods came an outgrowth of basic methods (i.e. ray-tracing, scan-line), terminology (i.e. hemi-cube, bump mapping),

[0] *Presented at Skigraph '99 – Western Computer Graphics Symposium, Mar 1999, Sunshine Village, Canada*

[1] In this paper, *visual simulations* are simulations with visual properties. They are not only simulations with visual interfaces, as commonly referred to by the simulation community.

and general tools (i.e. rayshade, optik). To add life to these images, many additional methods were introduced for animating visual attributes, such as shape, color, and location. The results of these additional methods have produced mixed results. Unlike what occurred for image production, these additional methods have not generated an strong outgrowth of basic methods, terminology, and general tools. Except for some basic methods, such as key-framing, interpolation, and motion capture, a common consensus of ideas and terms has not been be coined successfully. It is common for users to employ basic phrases for developing of a rendering system, not for developing an animation system. However, notable works in animation such as OBLIQ-3D [15], TBAG [4], and UGA [29], are initiating the drive to produce such terms.

The graphics community has not yet begun to use many of the latest ideas proffered by the simulation community for system development, such as multi-model modeling [7] and process abstraction [5]. These high-level techniques permit users to integrate many modalities into one system. Common themes, such resource allocation and contention [22], are also missing from the works of the graphics community. Algorithms for resource contention are useful to manage scenes with many competing objects. Also absent are methods of analysis and validation [14]. These methods are important to verify that the outputs of animated scenes are correct and valid.

2.2.2 Computer Simulation

Computer simulation seeks to create and to analyze programs that simulate the behaviors of real-world systems. The field offers a strong vocabulary of terms, methods, and tools to describe the benefits of various methods in simulation design. Some researchers, such as Ziegler [27], are also seeking to describe the field formally. They present formal notations that indicate how simulations act and what systems the simulations may emulate.

To provide greater aid in analysis and presentation [26], many tools from simulation, such as CINEMA[17], SIMGRAPHICS[21], and PROOF ANIMATION[3], provide elements to visual their output. Unlike their counterparts in graphics, most of these tools employ rudimentary techniques to visualize the properties of scenes and to illustrate the flow of information over time. Most approaches in simulation lack mechanisms to properly control the evolution of information. Many methods exist to control the execution of these mechanisms, but not what these mechanism produce as output. Graphics techniques, such as those used by ASA [20] and CONDOR [10], provide such useful methods.

3 Issues to Resolve

There are many difficult issues to resolve to successfully integrate computer simulation with computer graphics. Differing terminology (or lack of it), alternative aims, and dissimilar uses are a few of the many that make integration difficult. Many of these difficulties can be derived from three main issues, *software architecture*, *software coordination*, and *indeterminant maps*. The next three sections identify these issues and describe their relevance.

3.1 Software Architecture

A software architecture defines the organizational structure of a system. It examines structural issues, such as controls and protocols, and establishes corresponding ways to describe them[2]. In simple terms, a software architecture loosely defines the elements within a system, the relationships between the elements, and the form of communication that binds the elements.

The creation of a software architecture for visual simulation is vital for integrating computer graphics with computer simulation. The reasons are as follows:

- It would readily identifies the major (software) divisions of a visual simulation. This information would clarify how methods from graphics and simulation should be adapted for use in an integrate environment.

- It would facilitate that development of standards for system design. Consisting of common terms and methods, standards establish goals and constraints on integration. Without a software architecture, users are forced to integrate according to their own designs. Although this approach promotes great flexibility, it exposes the users to the pitfalls of *architectural mismatch*[8]. The users suffer if they realize a design for integration that conflicts with those introduced implicitly by the designs by other users.

- It supports a foundation upon which to manage the growth of complex systems. Many methods from both fields primarily address short term concerns, such as rapid prototyping. Successful integration of the two fields requires that each work properly in the long run. Fortunately, there is a growing trend in each field to provide such mechanisms. For example, simulation frameworks [16], composition techniques [6], and experimental frames [28, 24] introduce methods to describe simulations as the interplay of environments and interacting components, many which are potentially reusable and many which are useful for software integration.

3.2 Software Coordination

The intent of software coordination is to methodically organize the communications between interacting components. As components communicate, they exchange information and change state. The focus of software coordination is manage communications without realizing or disrupting the software elements that perform computational tasks. The foundations of software coordination appear in coordination theory [13], which advocates the separation of computation from coordination. Produced separately, the two tasks are easier to manage, comprehend, and reuse.

The use of software coordination for visual simulation is very helpful towards the integrate computer graphics with computer simulation. The reasons are as follows:

- It permits users to separately integrate the computational and coordinational tasks of computer simulation and computer graphics. Each task is combined separately from the other. After integration occurs, the two resultant solutions are put together to form a complete system.

- It permits users to quickly separate and substitute alternative tasks. Users select and replace the tasks that best suit their needs from either computer simulation or computer graphics, or both.

3.3 Indeterminant Maps

The integration of computer simulation and computer graphics is not only a software issue; therefore, it is neither an issue of computation nor coordination. It is also an issue of deciding how methods and visual information relate. Many methods in simulation do map easily onto changing visual attributes. For example, the command to add an element to a queue does not clearly indicate how it is perceived visually. Simply updating the position of a visual element onto a queue is not as easy as it appears. One must consider the amount of time that passes while the element joins the queue. In most simulation, this action is perceived as instantaneous. One must also consider the reaction of the other elements on the queue. They may or may not react to the element joining the queue. These types of scenarios, referred to as *indeterminant maps*, are common and frequently encountered by those studying software visualization [23]. A resolution to this mapping is vital to ensure that many existing algorithms from simulation integrate easily with that of computer graphics.

4 An Architecture for Visual Simulation

This section presents a (possible) software architecture for visual simulation by partitioning computer graphics and computer simulation into its main parts, and then, integrating the results. The results highly support to the use of software coordination to manage the flow of information between architectural parts.

4.1 Elements of a Visual Simulation

Both the computer graphics and computer simulation communities employ algorithms and techniques to create visual displays. Although they seek similar goals, each maintains an alternative decomposition of a visual simulation. Computer graphics dissects a visual simulation into three elements: *modeling*, *rendering*, and *animation*. Modeling produces virtual scenes by defines entities, such as geometric shapes and cameras. Rendering examines the entities to produce synthetic images. Animation alters the state of the entities over time. Computer simulation decomposes a visual simulation into four elements:[16]: *model creation*, *decision making*, *temporal management*, and *display building*. Model creation identifies the entities within a simulation that are of great interest. Entities represent tangible things, such as machines and parts, or intangible things, such as queues and routes. Decision making applies algorithms to evaluate or to alter the state of entities. Temporal management advances the state of simulated time. Display building visualizes entities for presentation and analysis.

Discrepancies between the two decompositions are understandable. The computer graphics community proposes a decomposition that mimics traditional animation. Elements serve to build and to animate virtual displays. Rendering visualizes what modeling creates and animation modifies. The simulation community proposes a decomposition that emphasizes the interaction between entities and the processes that control them. Elements serve primarily to establish and to observe interactions, not to create and to produce animated visuals.

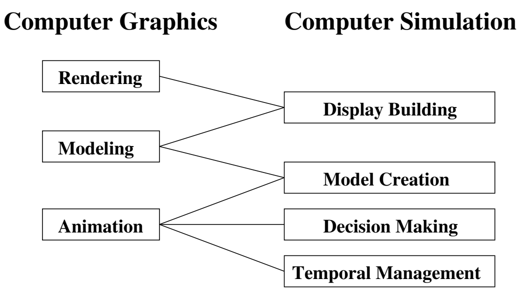


Figure 1: Relationships between Decomposition

Figure 1 illustrates the mapping between the two decompositions. Many of the elements from one community map to multiple element of the other community. This homomorphic mapping identifies an important division of concerns that one community recognizes and the other does not. Computer graphics identifies the elements within display building and model creation that render, model, and animate. Simulation identifies the elements within modeling and animation that display, create, decide, and manage. Collectively, the two decompositions produce a single decomposition that addresses the concerns of both communities. This single decomposition establishes the six elements of a visual simulation: *rendering*, *display modeling*, *model creation*, *animation modeling*, *decision making*, and *temporal management*. RENDERING visualizes what DISPLAY MODELING creates; ANIMATION MODELING animates what MODEL CREATION builds; and TEMPORAL MANAGEMENTS advances what DECISION MAKING decides.

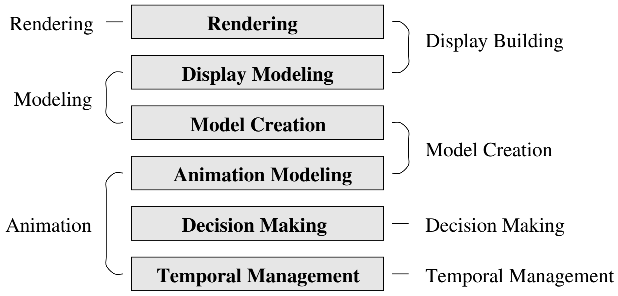


Figure 2: Elements of a Visual Simulation

Figure 2 identifies a mapping between the original elements and those of the collective decomposition. Original elements with fewer mappings, such as rendering of graphics and temporal management of simulation, offer greater contributions. They attack a smaller range of problems and establish few constraints that effect multiple elements. The overlap of "modeling" elements illustrates the difficulties of modeling a visual simulation and identifying its modeling components.

4.2 Element Relationships

The collective decomposition clarifies and unfolds many of the relations that exist in the original decompositions. Visual simulation is neither a simple interaction of three elements from computer graphics nor an elaborate collaboration of four elements from computer simulation. It is a combination of both. The collective decomposition organizes a visual simulation as a collection of algorithms that dynamically control the interaction between modeling, rendering, and animation. The state of a system changes as the algorithms decide when and how elements interact.

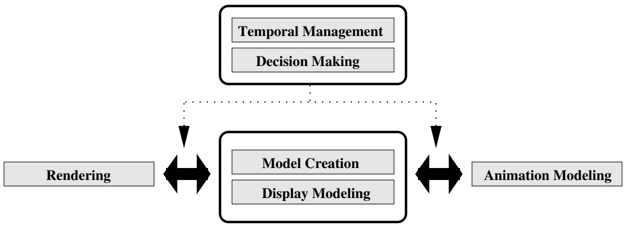


Figure 3: Relationships within a Visual Simulation

Figure 3 visualizes the elements and relations of the collective decomposition. The dark semi-rectangular frames identify elements with strong ties. MODEL CREATION accumulates DISPLAY MODELING structures to establish visual properties. DECISION MAKING collaborates with TEMPORAL ADVANCEMENT to specify dynamic interactions. The broad bidirectional arrows signify communication links. ANIMATION MODELING sends and receives information from MODEL CREATION to establish and to modify state. RENDERING interprets DISPLAY MODELING structures to synthesize virtual scenes. Emanating from the top frame, the speckled line specifies functionality. It indicates that the DECISION MAKING/TEMPORAL MODELING frame controls communication, not element states. The state of a system changes as the top frame manipulates the flow and exchange of information.

4.3 Element Communications

The collective decomposition identifies, not regulates the communication that occurs within a visual simulation. As seen in Figure 3, communication serves to update and to visualize the state of models. Neither the collective decomposition nor the original decompositions specify a protocol or plan for communication. Developing a general communication protocol for computer graphics and computer simulation is still an active area of research.

Although the collective decomposition does not explicitly define a protocol, it expresses concerns for one. It suggests that the protocol maintain element boundaries and that the protocol adapt to external influences. Keeping elements apart fosters rapid development and integration of prototypes. Elements with well defined boundaries are easier to identify and to extract. An adaptive protocol is fundamental for visual simulation. TEMPORAL/DECISION elements must coordinate interactions to establish time-varying behaviors.

5 Coordination for Visual Simulation

This section present a method of software coordination that is a proving of great use to visual simulation. This method, referred to as *interaction-based programming*, encourages a separation between computation and coordination. Users devise computational tasks which compute states separate from and coordinational tasks which regulate communications.

To show how this method works for visual simulation, this section introduces interaction-based programming, and then, progresses into an application of its use for each field. It concludes with some comments on the application of interaction-based programming to the integration of the two fields. The plan works under the assumption that if the method works for each field separately, then it should work for both together.

5.1 Interaction-Based Programming

In interaction-based programming, the computational tasks are provided by a set of software components that establish state and transform input to output. The coordination tasks are provided by a set of software interactions, first-class language constructs, that establish contexts for components to communicate. Each context describes a pattern of communication independent of the types of components they involve and of the computations which the components perform. The benefit of software interactions is that they localize the relationships among components that are typically spread across multiple statements of multiple components when using common programming language constructs, such as function calls. Because they are distinct, software interactions neither hampers nor obscures the code that computes state.

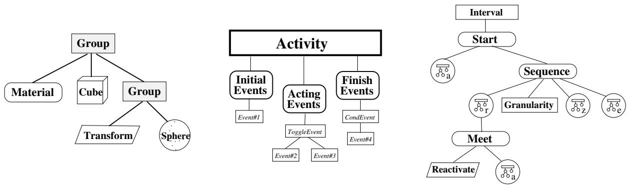
Interaction-based programming supports a strong foundation upon which to build time-varying systems, i.e. visual simulation. Coordination tasks readily support timing mechanisms, such as sequence and temporal intervals[1]. As simulated (or real) time advances, a time-based, run-time evaluator examines the timing mechanisms of the coordination tasks and evaluates those ready for activation. Automatic maintenance and execution of parallel events occurs as the evaluator parses active tasks concurrently. The evaluator ensures the validity of parallel events (i.e. state changes) by managing the tasks carefully. Parallel tasks that are found to conflicts raise errors.

5.2 Applied Separately

Interaction-based programming shows great promise for computer graphics and computer simulation. It serves as the foundation for IBAL, an interaction-based format for scene animation. [11] Virtual scenes update as interactions pass information between scene elements. For simulation, interaction-based programming guides the flow of information between elements that produce and capture data, such as queues and resources. This section begins with a quick recap of IBAL to illustrate the practical application of interaction-based programming to computer graphics, and then, presents some work with the application of interaction-based programming to computer simulation.

5.2.1 Computer Graphics

IBAL, an interaction-based animation language, establishes a general specification to describe the contents and behaviors of time-varying scenes. As illustrated in Figure 4, 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 actions that produce state changes. The time graph defines a temporal ordering of behaviors. It indicates how actions relate and execute over time.



(A) Scene Graph **(B) Action Graph** **(C) Time Graph**

Figure 4: IBAL Graphs

As can be viewed on the web[2], IBAL facilitates the production of a wide variety of scenes and encourages the reuse of graphical elements. However, it is not without limitation. 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 [27], 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.

5.2.2 Computer Simulation

Much of the work in computer simulation is built around the development and maintenance of data as it flows within a system. A common application in (traditional) simulation is managing the flow of

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

```
/* create resources */
Resource sink, source( 0, 480 ), resAct
/* activity requires resAct to work */
Activity inspect( exponential );
inspect.addRequirement( resAct );
/* queue uses the activity to process data */
FifoQueue queue;
queue.add( inspect );
/* establish connections */
source.addBranch( queue );
inspect.addBranch( finish );
```

Figure 5: Source to Sink via YANSL

information through a system involving source, sinks, and activities. The activities process and control the flow the information from sources to sinks. For interaction-based programming, these types of problems are relatively easy to manage and control. However, for many tools from simulation that use standard function-based approaches, such as SIM++ [12] and YANSL [9], these system can be troublesome. The final implementation (and its algorithm) produces valid results, but at the cost of less adaptability and the loss of dynamic control. These two drawbacks limit their versatility to model wide varieties of systems and to accept the integration of other techniques, such as those from computer graphics.

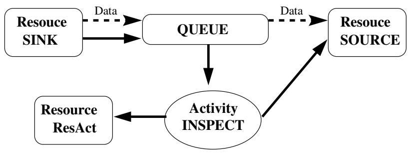


Figure 6: Simulation

The code in Figure 5 presents an example of a source to sink problem implemented with YANSL. As illustrated with the diagrams and solid lines of Figure 6, the code places a queue between the source and sink. Data enters the queue from the source and leaves the queues by way of the activity `inspect`. The activity works when resource `resAct` is available; otherwise, it suspends until the resource is free again. The dashed lines indicate the flow of information through the system. As can been seen from the functions calls in the code (shown as solid lines in the Figure), the function calls do not accurately reflect the flow of information that passes through the system. Therefore, when the code is partitioned or re-arranged, as can happen when it is integrated with other techniques, it becomes hard to understand the impact of the changes to the system and to the flow of data.

The code in Figure 7 recreates the same simulation with an interaction-based approach. The tasks consist of numerous soft-

```
/* create process (aka resource or activities) */
Process sink, inspect, sink, resAct;
/* form coordination tasks */
Task *send = queuing( src, queue );
Task *fini = dequeuing( queue, sink );
Task *link = linking( queue, inspect );
Task *wait = setResource( inspect, resAct );
/* apply temporal information */
Timing *doSend = send.set( 20, 40 );
Timing *doFini = fini.set( 20, 40 );
Timing *doLink = link.set( 20, 40 );
Timing *doWait = wait.set( 0, 100 );
```

Figure 7: Source to Sink via Software Interactions

```
Task* queuing( SRC, QUEUE ) {
    if SRC is READY and QUEUE is not full then
       PASS data from SRC to QUEUE
       inform SRC that data was passed
    endif
}

Task* dequeuing( QUEUE, SINK ) {
    if QUEUE is READY to EMIT data then
       PASS inform from SRC to QUEUE
       inform QUEUE that data was emitted
    endif
}

Task* linking( QUEUE, INSPECT ) {
    if QUEUE has waiting data then
       PASS data QUEUE to INSPECT
    if INSPECT has finished lookin at data then
       inform QUEUE to EMIT data
}
```

Figure 8: The Interactions of the Tasks in Figure 7

ware interactions that manage communications between the parts. As defined textually in Figure 8, the tasks control the flow of information between the processes. Each task regulates communications by determining if its constituents are ready; and once ready, the task passes data from once part to the next. The benefit of the tasks are that they accurately reflect the flow of data through the system and they readily accept temporal information. Unlike the previous code segment, this newly formed code segment can dynamically turn communications on and off. This adds the versatility of the definition and its ability to control the simulation with higher level abstractions.

5.3 Integration

The successful application of interaction-based programming to both computer graphics and computer simulation shows great promise to aid in the integration of the two fields. More work is necessary to determine if more simulation techniques adapt themselves easily to software coordination. If successful, future integration schemes may consist of set of templates and structures that setup interactions from graphics into sequences like that formed from simulation. More work is necessary to determine if this is easy to do and if it facilities easy programming.

6 Conclusions

This paper has presented an beginning to integrate computer simulation with computer graphics. It has present software architecture, software coordination, and indeterminant maps as three majors barriers to integration. Of these barriers, the papers present solutions for the first two, an architecture composed from an analysis of existing architectures, and interaction-based programming, an method of separating coordination from computation. Greater work is necessary to determine if these solutions can provide support to address the third barrier.

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] Paul C. Clements and Linda M. Northrop. Software architecture: An executive overview. Technical Report CMU/SEI-96-TR-003, Software Engineering Institute: Carnegie Mellon University, Pittsburgh, Pennsylvania, February 1996.

[3] Nancy J. Earle and James O. Henriksen. Proof animation: Reaching new heights in animation. In Jeffrey D. Tew, Mani S. Manivannan, Deborah A. Sadowski, and Andrew F. Seila, editors, *The 1994 Winter Simulation Conference*, pages 509–516, Lake Buena Vista, FL, Dec 11-14 1994. Institute of Electrical and Electronic Engineers.

[4] Conal Elliott, Greg Schechter, Ricky Yeung, and Salim Abi-Ezzi. Tbag: A high level framework for interactive, animated 3d graphics applications. In *Computer Graphics*, pages 421–434. ACM SIGGRAPH, Aug 1994.

[5] Paul A. Fishwick. The role of process abstraction in simulation. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 18–39, 1988.

[6] Paul A. Fishwick. SIMPACK : Getting started with simulation programming in C and C++. In *Winter Simulation Conference*, pages 154–162. ACM Association for Computing Machinery, Dec 1992.

[7] Paul A. Fishwick. *Simulation Model Design and Execution*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[8] David Garlan, Robert Allen, and John Ockerbloom. Architecture mistmatch or why it's hard to build systems out of existing parts. In *17th International Conference on Software Engineering*, pages 179–185, Seattle, WA, June 1995. IEEE.

[9] Jeffrey A. Joines and Stephen D. Roberts. Design of object-oriented simulations in c++. In John M. Charnes, Douglas J. Morrice, Dan T. Bunner, and James J. Swain, editors, *The 1996 Winter Simulation Conference*, pages 65–72. Institute of Electrical and Electronic Engineers, 1996.

[10] Micheal Kass. Condor: Constraint-based dataflow. In *Computer Graphics*, pages 321–330, Chicago, IL, July 1992. ACM SIGGRAPH.

[11] Gene S. Lee. A general specification for scene animation. In *International Symposium on Computer Graphics and Image Processing*, Rio De Janeiro, Brazil, October 1998. SIBGRAPI.

[12] Mark C. Little and Daniel L. McCue. Construction and use of a simulation in c++. Technical Report 437, University of Newcastle upon Tyne, 1993. Computer Science Tech Report.

[13] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.

[14] I. Mitrani. *Simulation Techniques for Discrete Event Systems*. The Cambridge University Press, Cambridge, England, 1982.

[15] Marc A. Najork and Marc H. Brown. Obliq-3d: A high-level, fast-turnaround 3d animation system. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):175–193, June 1995.

[16] David B. Pratt, Phillip A. Farrington, Chuda B. Basnet, Hernant C. Bhuskute, Manjunath Kamath, and Joe H. Mize. A framework for highly reusable simulation modeling: Separating physical, information, and control elements. In *The 24th Annual Simulation Symposium*, pages 254–261, New Orleans, Louisiana, 1991. IEEE Computer Society Press.

[17] David M. Profozich and David T. Sturrock. Introduction to siman/cinema. In Jeffrey D. Tew, Mani S. Manivannan, Deborah A. Sadowski, and Andrew F. Seila, editors, *The 1994 Winter Simulation Conference*, pages 427–430, Lake Buena Vista, FL, Dec 11-14 1994. Institute of Electrical and Electronic Engineers.

[18] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants.* Springer Verlag, 1991.

[19] W. T. Reeves. Particle systems: A technique for modeling a class of fuzzy objects. In *Computer Graphics*, pages 359–376. ACM Siggraph, July 1983.

[20] Craig Reynolds. Computer animation with scripts and actors. In *Computer Graphics*, pages 289–296. ACM SIGGRAPH, July 1982.

[21] Edward C. Russell. Simscript ii.5 and simgraphics tutorial. In Gerald W. Evans, Mansooreh Mollaghasemi, Edward C. Russell, and William E. Biles, editors, *The 1993 Winter Simulation Conference*, pages 223–227, New York, 1993. Institute of Electrical and Electronic Engineers.

[22] Herb Schwetman. Csim: A c-based, process-oriented simulation language. In James R. Wilson, James O. Henrikson, and Stephen D. Roberts, editors, *The 1986 Winter Simulation Conference*, pages 387–396, Washington, D.C., Dec 8-10 1986. Institute of Electrical and Electronic Engineers.

[23] John T. Stasko. Tango: A framework and system for algorithm animation. *IEEE Computer*, 23(9):27–39, September 1990.

[24] David T. Sturrock and C. Dennis Pegden. Introduction to siman. In Edward A. MacNair, Kenneth J. Musselman, and Phillip Heidelberger, editors, *The 1989 Winter Simulation Conference*, pages 129–139. Association of Computer Machinery, Association of Computer Machinery, 1989.

[25] Demetri Terzopoulos and Andrew Witkin. Physically-based models with rigid and deformable components. In *Graphics Interface '88*, pages 146–154, June 1988.

[26] Alexander Verbraeck and Gert-Jan de Vreede. Animation as a communication vehicle in simulation studies. In Alain Pave, editor, *Modeling and Simulation (ESM '93)*, pages 670–674, Lyon, France, 1993. The Society for Computer Simulation International.

[27] Bernard P. Zeigler. *Theory of Modelling and Simulation*. John Wiley & Sons, New York, 1976.

[28] Bernard P. Zeigler. Multifaceted, multiparadigm modeling perspectives: Tools for the 90's. In James R. Wilson, Stephen D. Roberts, and James O. Henriksen, editors, *The 1986 Winter Simulation Conference*, pages 708–712, ???, Dec 8-10 1986. Association of Computer Machinery, Association of Computer Machinery.

[29] Robert C. Zeleznik, D. Brookshire Conner, Matthias M. Wloka, Daniel G. Aliaga, Nathan T. Huang, Philip M. Hubbard, Brian Knep, Henry Kaufman, John F. Hughes, and Andries van Dam. An object-oriented framework for the integration of interactive animation techniques. In *Computer Graphics*, pages 105–111. ACM SIGGRAPH, 1991.