International Journal of CAD/CAM *Vol. 4, No. 1, pp. 19~32 (2004)*

INTERNATIONAL JOURNAL OF CAD/CAM
www.ijcc.org

3D Visualization of Discrete Event Simulation and Its Applications in Virtual Manufacturing

Yongmin Zhong* and Xiaobu Yuan[1]
Department of Mechanical Engineering, Monash University, Australia
[1]School of Computer Science, University of Windsor, Canada

Abstract – This paper presents a new approach to create 3D visualization from discrete simulation results. This approach connects discrete event simulation directly to 3D animation with its novel methods that analyze and convert discrete simulation results into animation events to trigger 3D animation. In addition, it constructs a 3D animation framework for the visualization of discrete simulation results. This framework supports the reuse of both the existing 3D animation objects and behavior components, and allows the rapid development of new 3D animation objects by users with no special knowledge in computer graphics. This approach has been implemented with the software component technology. As an application in virtual manufacturing, visualizations of an electronics assembly factory are also provided in the paper to demonstrate the performance of this new approach.

***Keywords*: Discrete event simulation, Discrete simulation events, Visualization, 3D animation, Software components and virtual manufacturing**

1. Introduction

Three-dimensional (3D) visualization of simulation results is an important and useful technique for engineering simulation [26]. It allows users to examine the complex processes of production plants in real-time and from different aspects. 3D visualization represents physical working environments with 3D graphics objects, and presents abstract simulation models by means of computer animation. It produces the visual presentation of what is to happen in the real world, thus providing users with a facility to study and analyze the production and logistics behaviors of industrial manufacturing operations. Through 3D visualization, users can obtain information not only from reports and statistical results, but also from the visual scenarios of the entire operational cycle within production plants, including supply chains, inventory, resource utilization, flow of materials, and the overall operation.

Despite its importance, 3D visualization of simulation results is still not widely accessible and its development cost is very large. The visualization of a stream of simulation results needs a connection between simulation and animation to convert discrete simulation events into animation-controlled events for direct stimulation of animation behaviors, and to convert simulation parameters and attributes into animation properties. Changes at one end of the connection should be automatically and immediately reflected at the other end. Furthermore, it requires a special animation framework to visualize in virtual environments the discrete events generated by simulation systems. Such framework should allow users with only average computer skills to easily and quickly create new animation scenarios and new visualization applications. Discrete simulation events provide only abstract information about state changes without providing details about how to perform an animation. It is therefore necessary for the animation framework to support both pure animation methods and the modeling of object behaviors. The practical nature of industrial applications also requires the animation framework to be capable of reacting to all incoming events in real time.

This paper presents a new approach for 3D visualization of discrete simulation results. This approach connects discrete event simulations with 3D animations, and supports rapid development of 3D animations from simulation results. It develops the methods to analyze discrete simulation results and to convert them into animation events for 3D animation. In addition, a 3D animation framework is constructed for the visualization of discrete simulation results. In the remainder of this paper, Section 2 gives a brief survey of the prior work related to this research. The analyzing and converting methods are then discussed in Section 3. After Section 4 outlines the animation framework, Section 5 presents system implementation and application results. Finally, Section 6 gives the conclusions and discusses future work.

*Corresponding author:
Tel: +61-(0)3-9905-1518
Fax: +61-(0)3-9905-1825
E-mail: Yongmin.Zhong@eng.monash.edu.au

2. Related Work

This brief survey first introduces the concept and progress of discrete event simulation. The focus then turns to the existing methods and practices in the visualization of discrete event simulation. Afterwards, current techniques for the rapid and cost-effective development of visualization systems are also discussed.

2.1. Discrete event simulation

Simulation is an important tool to predict real world phenomena through a computer model of a real phenomenon or system. It provides a way for decision-makers to evaluate different scenarios and identify the manufacturing system design that best meets their need. Most commercial software tools for engineering simulation are discrete event simulators. They simulate the behaviors of entities when an event occurs at a distinct point of time [11, 15, 22]. Events refer to the instants in time in which variable changes take place. They are called discrete simulation events that include all the state information of entities at an instant. The occurrence of events drives the simulation and the simulation clock. Nothing should happen between events. Therefore, time in a discrete event system does not proceed linearly but in irregular intervals [9]. An overview of discrete event simulation can be found in [1].

Discrete event simulation has been successfully used for the simulation and analysis of engineering applications in the past several decades. One important application is material flow simulation [16]. In this application, discrete simulators are used in the planning of material flow in a production system to arrange for effective and efficient production. Simulation is based on an abstract model of the plant that generates production reports and statistics. The process is time-consuming, and the generated results often do not match the expectations of experienced users or factory historical record. This mismatch places the usability of simulation systems in doubt, and causes hesitation in adopting them. In addition, the 2D simulation model of a complex system is only an abstract and simplification of the actual system [17]. The abstract simulation model is usually simplified to make it practical, but users may not be aware of the changes as these changes usually do not appear in the document of model specifications. Therefore, the verification and validation of the simulation model are still necessary. For example, Birta presented a knowledge-based method to validate the behaviors of discrete simulation models [2] and Gennart presented a concept for the validation of discrete event simulations based on recursively detecting and naming patterns of events [8].

Another problem of the event-oriented simulator is that its output is very difficult to understand. The interpretation of large amounts of digital information is suited for experts only. Users of such systems normally have neither the required knowledge nor the experience to study and analyze the simulation results. There have been continuous efforts to develop techniques for the presentation of abstract simulation models in visual formats so that non-expert users can understand and examine the underlying phenomena through visualization. 3D visualization is expected to enhance the users' confidence in engineering simulation, and to avoid potential errors caused by uncertainty and imprecision.

2.2. Visualization of discrete event simulation

For the visualization of discrete event simulation, a virtual factory was created by Kelsick [14]. It allows users to review simulation results by integrating the results of a discrete event simulator with a virtual factory model. The animation of the virtual factory is mainly based on the animation of each individual object in the virtual manufacturing cell. Every geometric model in the virtual factory is an entity in the program known as a node, and all nodes are structured in a node hierarchy. A child node in the hierarchy inherits the motion of its parent node. Nodes can be attached or detached depending on if the corresponding parts in the virtual factory are initially part of the node hierarchy. This is actually a method that uses scene graphs for geometry models. The animation of factory models is not automatic due to the missing knowledge needed to determine how to animate the complex behaviors of factory models.

Similar deficiency appears in the web-based simulation visualization tool developed by Salisbury *et al.* [28]. Coded with Java3D, this system allows users to observe simulation results remotely over the Internet by means of 3D animations. However, it provides no discussion on the complex behaviors of manufacturing. Kamat presented a system called Dynamic Construction Visualizer (DCV) for the visualization of construction simulations [12, 13]. This system obtains its input in the format of a Dynamic 3D Visualization Language, which consists of text instructions. The instructions describe basic behaviors such as creation (CREATE), placement (PLACE), movement (MOVE), and rotation (ROTATE) of 3D models in a browser. Since this language is not extendable, the system lacks flexibility when it has to deal with new simulation scenarios.

In the application of training, Dessouky developed a virtual factory teaching system [4]. It provides a workspace to illustrate the concepts of factory management and design of complex manufacturing systems. This system has the advantages of integrated simulations, engineering education, Internet connection, and virtual factories; but it still uses a separate simulation model from its graphical interface. Zhou integrated discrete event simulation and virtual reality technology in a system for the visualization of a manufacturing factory [36]. In this system, an animation script constructed with the data from a simulated trace file drives the movements of dynamically moving objects, such as materials, pallets, components, and transport vehicles. Interactive behaviors are added to each scene for better understanding of the manufacturing process and logistic operations. Unfortunately, animation scripts are for computer programmers. They are not suitable for manufactory workers, and are definitely not for decision makers. This system also has difficulty for animating large scale and complex manufacturing behaviors because of the limits of animation scripts. Another integrated method of virtual factory was presented in [35]. This method uses static simulation to evaluate factory layout, and dynamic simulation to evaluate the feasibility of operation plans. It can only deal with simple assembly lines, and cannot be extended to evaluate complex production processes.

There are a few discrete simulation packages, such as eM-Plant [6] and WitnessVR [34], which can directly generate the 3D animation from a discrete event simulation. However, the 3D models provided by these systems are not the real models in manufacturing processes, and therefore they do not reflect the true manufacturing scenarios. A few other packages, such as AutoMod [27] and Quest [23], provide 3D animation functions for simulating a manufacturing process. However, they do not provide any connections with discrete event simulations and users need to do many tedious menu interactions to define complex manufacturing behaviors. In addition, the 3D models provided by these are limited in some specific manufacturing application.

2.3. Rapid development of 3D visualization from discrete simulation

The cost of system development and the expandability of visualization systems are also topics under active investigation. There have been some approaches focusing on reducing the size and complexity of 3D visualization systems. They use reusable geometric objects in their animation processes. For instance, Luckas proposed to use animation elements in object-oriented system frameworks [18], and discussed some ideas of using object-oriented technology in 3D visualization [19]. In the method developed by Elcacho for the rapid generation of animation elements, animation elements encompass geometric description of their visual appearances and adaptable animation behaviors [5]. This work uses object-oriented methodology in the design of animation elements, but the emphasis is on the reusability enabled by overloading behavior methods. Similarly, Sun applied object-oriented methodology in the environment construction of a virtual shop floor [32]. In this exercise, solid models, behavior models, and control models are constructed into basic classes for easier extension. However, it does not provide discussions on how to construct and implement basic classes. Mueller-Wittig introduced a system for the visualization of electronics assembly [20]. This work is at its initial stage, and only the conceptual framework on visualizing electronics assembly process is available.

Although object-oriented approaches facilitate the reusability of objects and reduce development time, they do not explicitly address the designing of reusable frameworks while component-based software development does [21]. Component-based software development enhances software reuse and rapid development with its framework. Blanchebarbe developed a component-based 3D framework for the configuration of products [3]. Software components, rather than software objects, are used to encapsulate visual appearances and animation behaviors. The framework supports the reuse of components and primitive components can be visually composed into large scenes. Unfortunately, the composition of complex behaviors is not addressed. Quick presented a component-based framework for the visualization of simulation results [24], and discussed the monitoring and control of the visualization system [25]. This research, however, has not produced any convincing results but some conceptual framework and some ideas on visualization of simulation results.

Although object-oriented or component-based approaches reduce the effort on behavior modeling and animation programming, their supports are still weak in either the integration of simulation data into animation scenes or the connection of simulation results with animations. Wenzel presented an approach for mapping discrete event simulation models onto animation models [33]. This approach translates discrete simulation events into animation actions according to some predefined translation rules. However, no substantial results can be found to verify the feasibility of this method. Furthermore, this method cannot be extensively used since these rules are related to the specific application.

Different from other approaches, this paper develops the methods to analyze discrete simulation events and to convert them into animation events so that the direct connection from discrete simulation results to animations becomes possible for the first time. The incorporation of transformations in animation objects facilitates the process of behavior creation and editing. The adaptation of the techniques of 3D animation framework and 3D components further improves the development speed and stability of 3D application.

3. Analysis and Conversion of Discrete Simulation Events

Discrete simulation events produced by a discrete simulator cannot directly trigger animation sequences mainly because the mapping from discrete simulation events to animation is a many-to-one relationship. This section develops the methods to analyze discrete simulation events and to convert them into animation events for 3D visualization.

3.1. Analysis of discrete simulation events

Most of the simulation packages are capable of automatically producing a trace file. This trace file records events taking place during the time period of each simulation run. It is a text file including millions of records that describe the whole procedure of simulation. For example, listed in Table 1 is a segment of the trace file produced by the simulation tool "eM-Plant" [6] in its procedure of simulating an electronics assembly process. This table has four columns. The elements in the first column are the events that trigger the actions. The elements in the next two columns list the time when events happen and the material ID numbers respectively. The elements in the last column are the locations of materials in a format of "Factory-ID.Factory-Area-ID.Factory-Family-ID.Factory-Module-ID.Factory-Station-ID.*". In this format, symbol " * " indicates the last string of the location, which can be a product on a line ("Line"), a product on a station ("stn"), or an additional new product to a factory family ("SPsource"). Fig. 1 illustrates some special strings in the trace file by mapping these strings to 3D objects.

The behaviors of a particular material can be collected from this table. For instance, Table 2 contains a collection of items from Table 1 that shows some behaviors of the material P21:1.

The trace file in Table 1 and Table 2 shows that the discrete simulation events generated by simulation output provide the behavior information of the assembly process. However, the trace file is abstract and very large in volume. It is impossible for a non-expert user or even an expert to either establish the relationships between different discrete events or understand the complex manufacturing process. To display the simulation results, a visualization system has to convert discrete events into a special type of data that can trigger animation behaviors. One important task is to analyze the structure of discrete simulation events, and to display discrete simulation events in forms of a graph. The analysis process first extracts all the distinct event types and all the behavior information of every source object, and

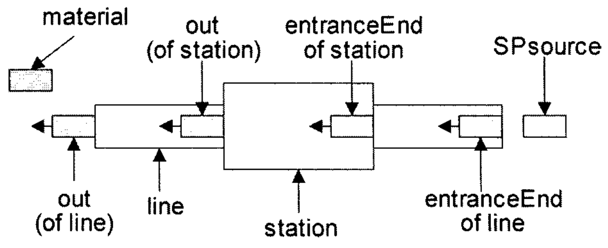

Fig. 1. Mapping the trace file to 3D objects.

Table 1. A small simulation trace script fragment

Event	Time	Material	Location
Out	1:00:16:17.0000	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
Out	1:00:17:03.2223	P21:2	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.stn
Out	1:00:17:03.2223	P21:3	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.Line
Out	1:00:17:03.2223	P21:7	PRODN.FATP2.FATP2SA.JSPmod.SPsource
Out	1:00:17:03.2223	P21:8	PRODN.FATP2.FATP2SA.JSPmod.SPsource
entranceEnd	1:00:17:05.2223	P21:2	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
Out	1:00:17:05.2223	P21:4	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.Line
entranceEnd	1:00:17:05.2223	P21:7	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.Line
Out	1:00:17:15.2223	P21:2	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
Out	1:00:17:21.8825	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.stn
Out	1:00:17:21.8825	P21:2	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
entranceEnd	1:00:17:23.8825	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.Line
Out	1:00:17:36.8825	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.Line
Out	1:00:17:56.4935	P21:3	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.stn
Out	1:00:17:56.4935	P21:4	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.Line
Out	1:00:17:56.4935	P21:8	PRODN.FATP2.FATP2SA.JSPmod.SPsource
Out	1:00:17:56.4935	P21:9	PRODN.FATP2.FATP2SA.JSPmod.SPsource
entranceEnd	1:00:17:58.4935	P21:3	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
Out	1:00:17:58.4935	P21:5	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.Line
entranceEnd	1:00:17:58.4935	P21:8	PRODN.FATP2.FATP2SA.STN_FATP2SA_01.Line
Out	1:00:18:08.4935	P21:3	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
Out	1:00:18:28.1026	P21:2	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.stn
Out	1:00:18:28.1026	P21:3	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
entranceEnd	1:00:18:30.1026	P21:2	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.Line
Out	1:00:18:30.7915	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.stn
entranceEnd	1:00:18:32.7915	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_04.Line
Out	1:00:18:42.7915	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_04.Line

Table 2. The behaviors of the material "P21:1"

Event	Time	Material	Location
Out	1:00:16:17.0000	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.Line
Out	1:00:17:21.8825	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.stn
entranceEnd	1:00:17:23.8825	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.Line
Out	1:00:17:36.8825	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.Line
Out	1:00:18:30.7915	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.stn
entranceEnd	1:00:18:32.7915	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_04.Line
Out	1:00:18:42.7915	P21:1	PRODN.FATP2.FATP2SA.STN_FATP2SA_04.Line

Table 3. An animation event

Material	Start Event	Start Time	Start Location	End Event	End Time	End Location
P21:1	Out	1:00:17:21.8825	PRODN.FATP2.FATP2SA.STN_FATP2SA_02.stn	Out	1:00:18:30.7915	PRODN.FATP2.FATP2SA.STN_FATP2SA_03.stn

then describes the complex manufacturing process in a graph structure.

This graph structure reveals all possible sequences of events. It helps users to understand the complex process, and to create the logical link between discrete event simulation and animation. Since the source of all the events comes from different projects, these discrete events often have different formats. Therefore, discrete events need to be normalized before they can be processed further. Furthermore, a routing mechanism also has to be implemented to ensure that the discrete simulation events can be directed to the animation objects on which they have an effect.

In most cases, the discrete simulation events cannot directly control the animation since discrete simulation events and animations do not have a one-to-one relationship. In fact, the relationship is a many-to-one mapping. For example, in Table 2, the discrete simulation events at the time instances of "1:00:17:21.8825", "1:00:17:23.8825", "1:00:17:36.8825", and "1:00:18:30.7915" can all map to an animation event shown in Table 3. As a result, the discrete simulation events have to be converted into animation events by means of translation. In addition, discrete simulation events often lack necessary information when generating animation events because discrete simulation events only provide the state information of materials at different events/time. Additional data has to be acquired from other sources to decide the parameters needed before calling an animation behavior function. In comparison to discrete simulation events, this data is static and needs only to be acquired once.

3.2. Conversion of discrete simulation events into animation events

It is therefore very important to build proper correspondence between the discrete event simulator and the animation scene. As shown in Fig. 2, this vital task is accomplished in the proposed framework by an analyzer that analyzes discrete simulation events, and translates them into animation events by transforming a sequence of two and more successive discrete simulation events into one animation event. To reflect the manufacturing process described in the simulation output and to aid users creating animation events, a graph structure is built to show all possible successor events of each event. This graph can be edited by skipping unimportant events. Upon the completion of the graph structure, users can pick out sequences from the graph and generate corresponding animation events. Types of animation events can also be defined in the analyzer. An animation event is generated after the occurrence of a certain sequence of one or more discrete simulation events. Each animation event provides a set of parameters. These parameters are calculated from a set of sequential discrete simulation events.

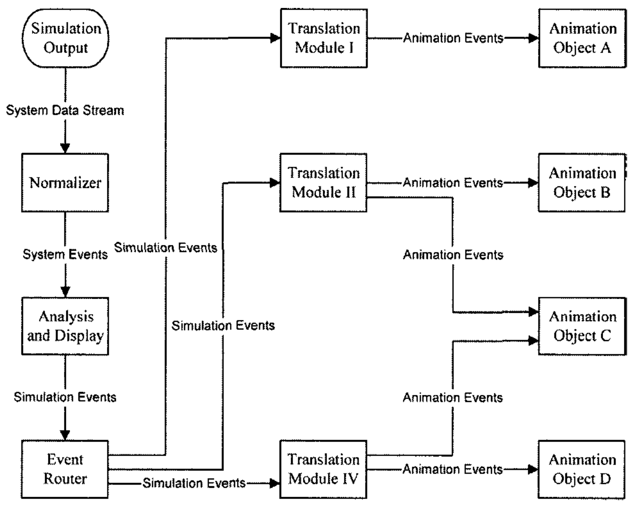

Fig. 2. Analyzer framework.

Before the analysis takes place, however, a normalizer is necessary to convert the data input stream into system events. Regardless of their sources, all system events share some common characteristics. Each event consists of a set of data fields. In addition, events provide an event type and a source object ID. The source object is the object to which the system event is related. All normalizer modules are implementations of the same interface. They are placed in a special package. Users can choose the normalizer type from all classes contained in the package at run time. To add a new implementation to the normalizer, a user simply places the file of this new class into the package. There is no need to change the main program.

The responsibility of the analysis and display module in the analyzer framework is to establish the relationships between different discrete event types according to the ID of source objects and their sequences of occurrence. Discrete simulation events are then produced from the discrete events and the relationships between the discrete events. The presentation of discrete simulation events is in the format of a graph structure, which reflects the assembly process and helps users to convert discrete simulation events into animation events. Since only a subset of event types needs to be translated, the event router module is employed to prevent other discrete simulation events from being sent to translation modules. It is in charge of dispatching incoming events to only those translation modules that request to receive them.

These discrete simulation events signal the change of an object position in the simulation environment. To trigger the animation of an object and the animation of other objects with which this object has interactions, the discrete simulation events are translated into animation events, and then sent to the animation objects whose animation behaviors are under the control of these events. The conversion of discrete simulation events is accomplished by a set of translation modules. The translation modules can be customized to the requirements of a specific project. They also have links to animation objects, which receive the created animation events during visualization. In cases when multiple modules are used, each event should only be sent to the modules that are responsible for its conversion to save system resources.

The translation modules work as a group of finite automata. They receive from the event router all the event types that appear in the input channel. For each event source of discrete simulation events that reaches the translation module, a new state is stored. State changes take place according to the translation rules of the automata. An animation event is created when a state change happens to be an end event. Its parameters are determined by the discrete simulation events received by the automata. The created animation event is finally broadcasted to all the animation objects, which are event listeners registered with the translation module for the specific event type.

4. Component-based Animation Framework

According to component theory, a software component is a building block in a software system that has an explicitly defined interface designed for reusability [29]. The proposed animation framework follows the object-oriented application frameworks [7]. It uses dedicated authoring tools to fill a generic skeleton application with components. The skeleton is in charge of invoking and integrating the components' functionalities. In this approach, a component can be selected from a library, and reused after minor adaptation and customization. Pre-defined patterns then determine the proper layout of components in the skeleton application. Since authors do not need to know the cooperation of components, no programming skills are required.

As shown in Fig. 3, the component-based animation framework consists of a 3D runtime framework and an authoring framework. The 3D runtime framework has the basic interfaces for loading the components. A 3D light, a 3D view, and an animation event can be plugged in this framework as components. All the animation objects are represented as 3D components and stored in the animation object library. The animation framework loads animation objects from the animation object library. While the rendering kernel provides the graphics functions to set up a scene and to render picture frames, the event scheduler provides the communication functions to handle all events, such as animation events, component events, and user interactive events.

In comparison, the authoring framework has an animation viewer to display the visualization results. It provides the visual tool for the author to configure 3D scenes, and to control the generation and release of animation events for 3D visualization. In addition, it has a behavior editor for the author to create new 3D animation objects, and to define their behaviors with the existing animation methods and the composition of 3D object models. The layout editor produces the scene model according to the layout of animation objects. It

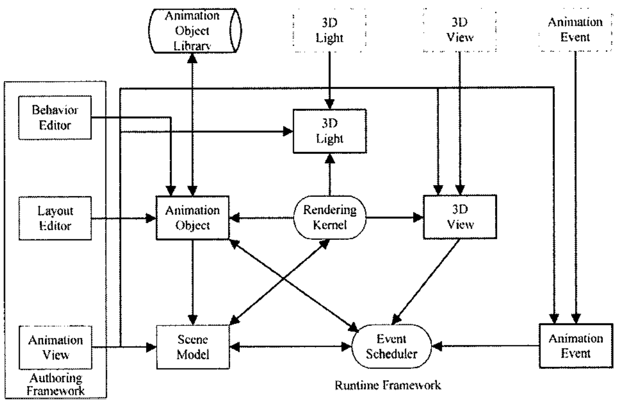

Fig. 3. Component-based animation framework.

is also responsible for connecting animation events to the related animation objects.

4.1. Animation objects

An animation object is a component with encapsulated geometry and behaviors. As shown in Fig. 4, the component framework uses a scene graph to store the geometric properties of an animation object, and encloses a group of animation methods to define its behaviors. The basic units of a scene graph are scene objects, and a scene object may contain other scene objects as its children. The scene graph is scalable in size and flexible in making changes to scene objects. The animation methods, on the other hand, define actions in response to events, each of which can be a modification to transformation, an interaction from the user, or a change of internal state. In addition, the framework employs three interfaces to establish the connection between events, behaviors, and animation of an animation object.

The event interface consists of an event handler and an event listener. The former is in charge of attribute accessing and event processing to fire, receive, and sort events; and the latter provides methods to link an event to a behavior. The event interface also includes some other basic features of a component, including component reflection and persistence. In comparison, the behavior interface provides access to the behavior description and other aspects of the component. The animation methods are represented as behavior components. The existing animation methods can be edited and new animation methods can be added through this interface. The animation methods are further implemented by animation interpolators at run time to provide the specific transformation information. The major animation interpolators are listed in Table 4, and they are for translation, rotation, translation with rotation, and scene graph location change of animation objects respectively.

The component framework uses a scene location node (SLN) to store the location of a scene object in the graph, and a transformation node (TN) to maintain

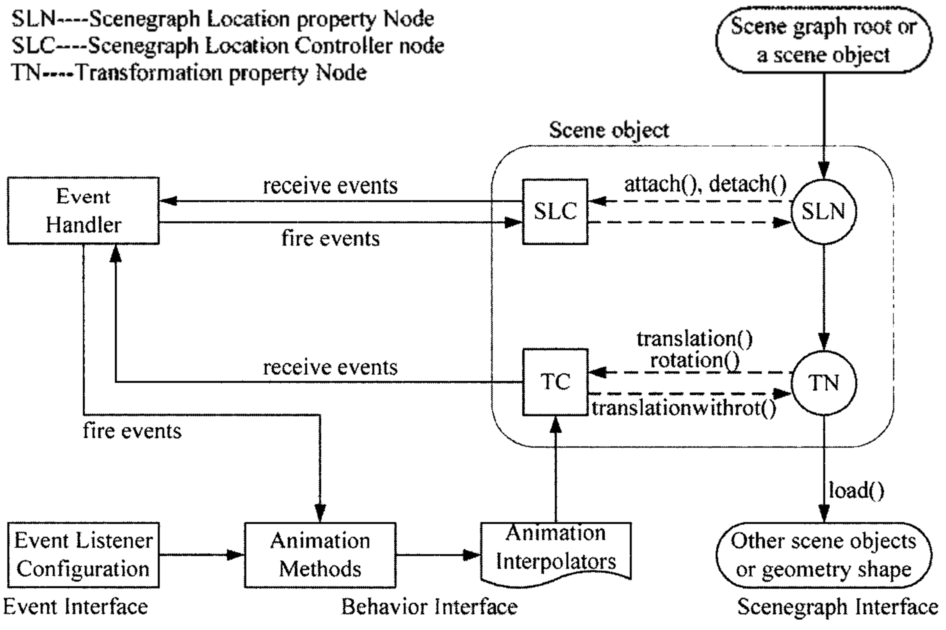

Fig. 4. The component structure of an animation object.

Table 4. Animation interpolators

Interpolator Constructors	Functions
Translation Interpolator (long t1, long t2, Transform Group tg, Transform 3D t3d, Point 3D p1, Point 3D p2);	To interpolate a translation of an object from location A at the time T_1 to location B at the time T_2
Rotation Interpolator (long t1, long t2, Transform Group tg, Transform 3D t3d, Quat 4D q1, Quat 4D q2);	To interpolate a rotation of an object about an axis from angle A at the time T_1 to angle B at the time T_2
Trans With Rot Interpolator (long t1, long t2, Transform Group tg, Transform 3D t3d, Point 3D p1, Quat 4D q1, Point 3d p2, Quat 4D q2);	To interpolate a translation of an object from location A at the time T_1 to location B at the time T_2 while a rotation about the axis of the object from angle A at the time T_1 to angle B at the time T_2
Scene Graph Location Interpolator (long t, Branch Group parent, boolean create);	To remove and attach an animation object from one scene graph branch to another scene graph branch

the transformation values of the sub-graph attached to it. In addition, two controllers are used. One is a transformation controller (TC) that updates transformation at run time according to the information provided by animation interpolators. Meanwhile, it fires an event to notify an update to the transformation. The other is a scene location controller (SLC) that controls both the detachment and attachment of a scene object from and to a scene graph branch according to the receiving events. Meanwhile, it fires an event after the attachment or detachment. The goal of this controller is to facilitate dynamic scene graph structure and to generate an instant impact on the transformation matrix of scene objects.

The scene graph interface provides access to the scene graph with its three types of methods. The first type works with the root scene graph. It includes methods to attach a scene object to or detach a scene object from the scene graph freely at run time, which promotes the reuse of scene objects. The second type work with the sub-scene graph. It includes methods to set the geometry of a scene object or link to other scene objects. The third type includes methods to update time-varying properties in real time.

Furthermore, an event-driven animation thread can also be easily and quickly generated to handle the animation execution. The animation thread consists of a series of SLC and TC controllers in the time order of the occurrences of the events corresponding to the controllers. A behavior can be finally decomposed into a series of transformations under the control of controllers. The execution of a behavior involves a submission of all controllers in the animation thread of this behavior, and the initiation of the controllers one by one in the time order of the occurrences of the events corresponding to the controllers.

4.2. Animation object library

The presented animation framework supports the creation of 3D animation objects through its 3D animation object library. Stored in the repository of this library are animation objects, each of which is ready to use in a drag-and-drop fashion. Shown in Fig. 5 are some of the animation objects it has for electronics assembly, with their animation methods listed Table 5.

Based upon the Java Beans component model, this library supports rapid development of new objects. It allows users to compose new objects from the existing objects or from parts of the existing objects. For newly created animation objects, users can compose new animation methods from the existing animation methods. Either of the compositions does not require users to have any prior knowledge of internal implementation.

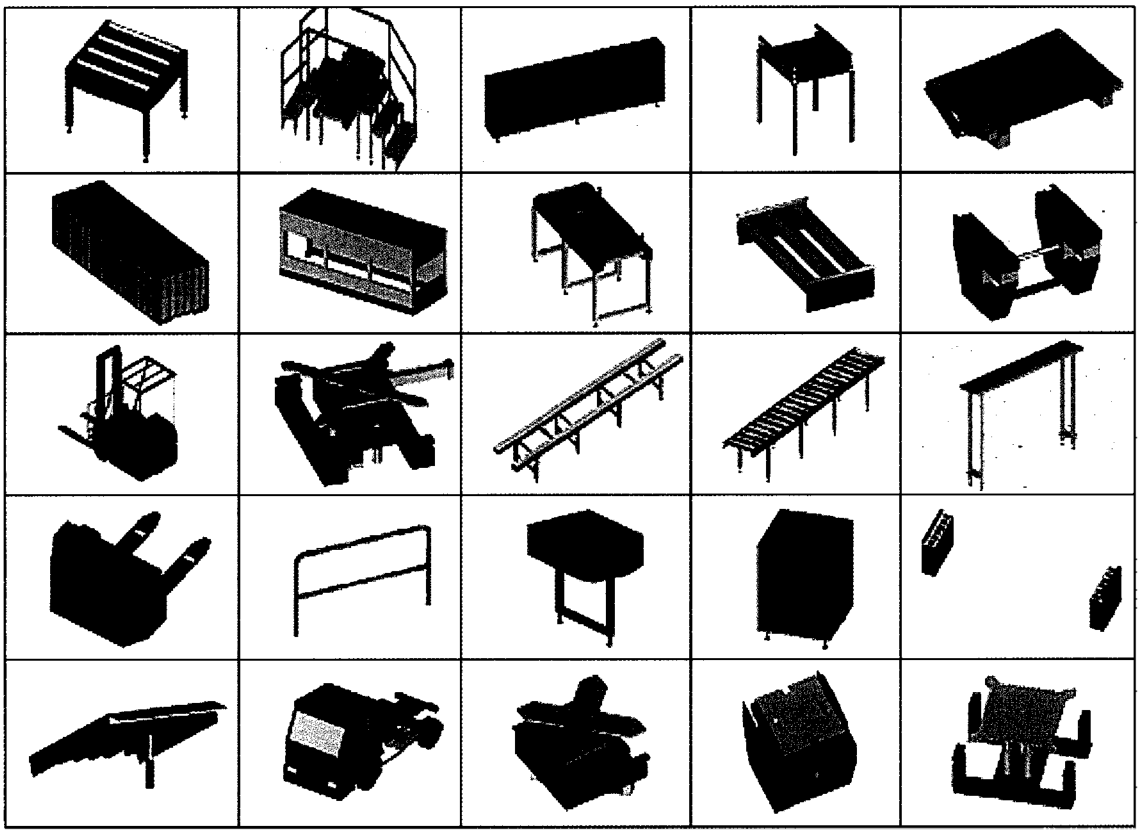

Fig. 5. Library of 3D factory elements.

Table 5. Some animation methods for animation objects

Animation Elements	Animation Methods	Actions

Crane	void **slideForkMoveLeft**(Long t_0)
	void **slideForkMoveMiddle**(Long t_0)
	void **slideForkMoveRight**(Long t_0)
	void **forkbaseMovePos**(Long t_0, float relpos);
	void **craneMoveForward**(Long t_0, Long t_1, float distance);
	void **craneMoveBackward**(Long t_0, Long t_1, float distance);
		slideFork move left
		slideFork move middle
		slideFork move right
		forkbase move up
		forkbase move down
		crane move forward
		crane move backward

primeMover	void **movePMoverForward**(Long t_0, Long t_1, float distance);
	void **movePMoverBackward**(Long t_0, Long t_1, float distance);
	void **turnPMoverRight**(Long t_0, Long t_1);
	void **turnPMoverLeft**(Long t_0, Long t_1);
		primeMover move forward
		primeMover move backward
		primeMover right turn
		primeMover left turn

Container Carrier	void **moveCCarrierForward**(Long t_0, Long t_1, float distance);
	void **moveCCarrierBackward**(Long t_0, Long t_1, float distance);
	void **turnCCarrierRight**(Long t_0, Long t_1);
	void **turnCCarrierLeft**(Long t_0, Long t_1);
		ContainerCarrier move forward
		ContainerCarrier move backward
		ContainerCarrier turn right
		ContainerCarrier turn left

bayPlatform	void **moveBayPlattformUp**(Long t_0, Long t_1);
	void **moveBayPlattformDown**(Long t_0, Long t_1);
		bayPlatform move up 45°

elevator	void **openDoor**(Long t_0);
	void **closeDoor**(Long t_0);
		elevator door slide
		open/close

forklift	void **moveForkliftForward**(Long t_0, Long t_1, float distance);
	void **moveForkliftBackward**(Long t_0, Long t_1, float distance);
	void **turnForkliftAround**(Long t_0, Long t_1);
	void **turnForkliftRight**(Long t_0, Long t_1);
	void **turnForkliftLeft**(Long t_0, Long t_1);
	void **moveBlade**(Long t_0, Long t_1, float relpos);
		forklift move forward
		forklift move backward
		forklift turn 180°
		forklift right turn
		forklift left turn
		blade move up
		blade move down

Automated Palletjack	void **movePalletjackForward**(Long t_0, Long t_1, float distance);
	void **movePalletjackBackward**(Long t_0, Long t_1, float distance);
	void **moveBlade**(Long t_0, Long t_1, float relpos);
		automated pallet jack move forward
		automated pallet jack move backward
		blade move up
		blade move down

Screwdriver	**screwObject**(Long t_0, Long t_1, Aelement *elemPtr);	ScrewObject
Lifter	**lifterUp**(Long t_0, Long t_1, Aelement *elemPtr, float startht, float endht); **lifterUp**(Long t_0, Long t_1, float startht, float endht); **lifterDown**(Long t_0, Long t_1, Aelement *elemPtr, float startht, float endht); **lifterDown**(Long t_0, Long t_1, float startht, float endht);	LifterUp lifterDown

Furthermore, the communication between components provided by the component framework enables this library to easily connect the animation functions with animation event sources. These sources control the object animations in the scene. They initiate animation events for them to trigger the corresponding animation functions.

The representation of a factory and the description of its inherent processes are complex in nature. They often result in a scene with over millions of polygons to render. To make visualization adaptive and capable of producing high-quality results on low-cost platforms, the animation framework has been established based on the concept of level-of-detail (LOD) in its visualization. The geometric model of each object contains three levels of detail in respect to the model's complexity, i.e. simple, moderate, and high. Fig. 6 shows the different LODs on a box sealer animation element. In this particular animation element, the low representation has approximately 400 polygons, the medium has approximately 1000 polygons, and the high has approximately 5000 polygons.

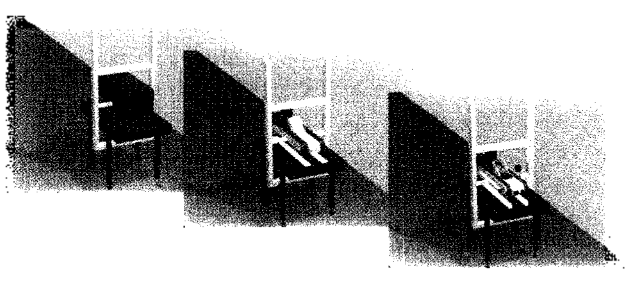
Fig. 6. Case sealing system (level of detail).

4.3. Authoring tools

The authoring framework provides tools to create or modify a 3D animation quickly and efficiently for different applications. These authoring tools include layout editor, behavior editor, and animation viewer. The layout editor is to visually construct a scene model from individual objects. With this editor, objects are selected from the 3D animation object library, and are positioned within the scene. All objects are organized into a tree hierarchy, which makes it easy to change the layout of scene subsets. Object customization is accomplished by accessing the public methods of objects, and object positions are defined either visually by drag-and-drop or manually by inputting the coordinates. The data to set the coordinate values are retrieved from the simulation specification. Any change to the simulation specification is immediately reflected in the positioning of objects in animation scene, and any change to object positions results immediate changes in simulation specification. Moreover, animation methods of objects can be connected to the animation events created by the analyzer.

Animation behavior editor is responsible for the creation and testing of new 3D animation objects to make the creation process of 3D animation objects faster and easier. This editor loads 3D objects for visualization. Animation behaviors are specified by customizing objects' behavior components. With this editor, new objects can be created from the existing 3D objects in the library, and new behavior components can be defined from combinations of the existing behavior components. A 3D object becomes an animation object with animation methods, and those methods are customized from behavior components. Furthermore, the functionality of the animation objects can be tested during the entire customization process. By calling the animation methods with test parameters, the visualization system is able to show the resulting animations immediately.

The animation viewer is to display the visualization result. To run an animation sequence for the first time, however, the animation viewer needs to control the generation and release of the animation events. After the animation objects receive the animation events, they start to generate animations by directly manipulating the scene's geometry and appearance in a fixed time frame. To enable users to search for any particular process events, the speed of animation is adjustable. Fast-forward and jumping to certain time points are allowed.

5. Implementations and Applications in Virtual Manufacturing

5.1. Implementations

A 3D visualization system has been successfully developed with Java3D and Java Beans. The implementation relied heavily on the software component technology. Java3D [31] uses a high-level 3D graphics kernel based on the concept of scene graph, and is ready to comply with component standards. The visualization system uses Java Beans [10] in its implementation, and has achieved flexible configuration and reusable building blocks, while keeping the complexity of this task low. The deployment of component technology allows easy integration with the existing visual building tools in both the initial system development phase and later for implementations in different applications.

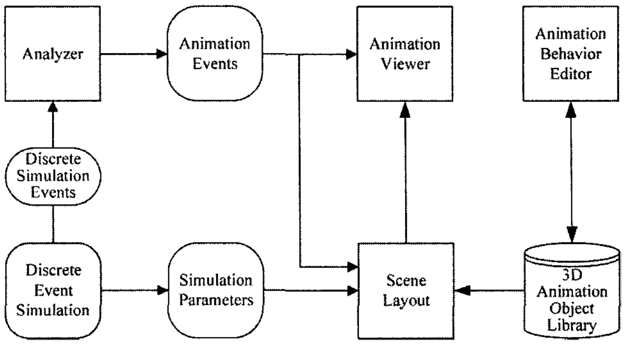
Analyzer
Animation Events
Animation Viewer
Animation Behavior Editor
Discrete Simulation Events
Discrete Event Simulation
Simulation Parameters
Scene Layout
3D Animation Object Library

Fig. 7. System flowchart.

The system flowchart is shown in Fig. 7. The discrete simulation produces discrete simulation events. The analyzer receives the discrete simulation events from the simulation, and translates the discrete simulation events into animation events. New animated objects can be created in the animation behavior editor from the existing 3D object models and animation methods. All 3D animation objects are stored in a library. In the scene layout editor, 3D animated objects from the library are laid out in a scene. Simulation parameters that have influence on the scene layout or its animation behavior are linked to the relevant objects. The connections between animation event sources and animation methods of the 3D animation objects can also be established. Upon the completion of the layout, the animation viewer displays the scene according to the animation events received from the analyzer.

5.2. Applications in virtual manufacturing

Virtual manufacturing is the use of computer models and simulations of manufacturing processes to aid in the design and production of manufactured products and environments [30]. It has been recognized as a desirable feature in modern manufacturing systems to improve product quality, increase diversity of products, and reduce product development time and cost.

The established approach for 3D visualization of discrete event simulation has been successfully applied in virtual manufacturing to crate a 3D simulation model of an electronics assembly factory. The process of constructing the virtual factory is shown as in Fig. 8. The manufacturing process information is collected from the real manufacturing system and is organized in different tables to describe the manufacturing process. According to the manufacturing process description, the 2D simulation model of the manufacturing system

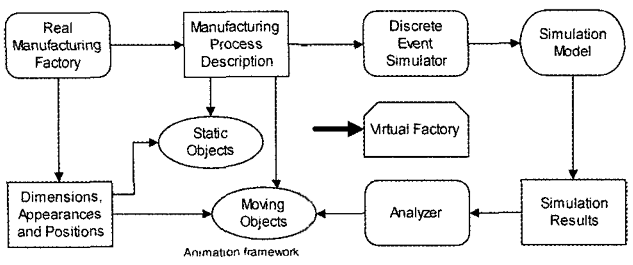

Fig. 8. Architecture of constructing a virtual factory.

is created by a commercial discrete simulation tool "eM-Plant" [6]. The simulation model is executed and generates results and reports based on the simulation run.

To visualize the 2D discrete event simulation of the manufacturing factory, the simulation results are transformed into animation events that can trigger and control the behaviors of the manufacturing system, and 3D objects are modeled according to their geometric dimensions and appearances, and are stored in the library. These 3D objects can be divided into two types: static objects and moving objects. The static objects refer to the objects with the fixed positions during the manufacturing process, such as building, production equipments and workstations, and production lines. Some objects whose positions are fixed during the manufacturing process while may be changed with the different kinds of products are also belonged to static objects. The quantity of each kind of static objects can be obtained from the manufacturing system description tables. The positions of the static objects can be obtained from the shop floor blueprint or measured from the actual shop floor. According to these positions, the environment (scene) of the manufacturing system is produced by the layout functions provided by the animation framework.

The quantity and positions of the moving objects such as materials, pallets, components, lifters and transport vehicles are obtained from the manufacturing process description tables. The movements of moving objects are triggered by the animation events provided by the analyzer.

With the created factory model, users can visualize in the 3D environment all the manufacturing processes, including the flow of materials, size of buffers, and line balancing. Based on the visualization results, refinements to the existing processes are made, for example, on minimal process time, reduced wait time, and reduced buffer size.

Demonstrated in Fig. 9 are the analysis and conversion of simulation results generated by "eM-Plant" for the electronics assembly process. The graph describes the structure and relationships of the discrete simulation events. The dialogue box shows the selected discrete simulation events for creating an animation event. The user can select a node in the graph and set it as the start event, end event, or middle event for creating an animation event.

In the following figures, Fig. 10 illustrates the layout

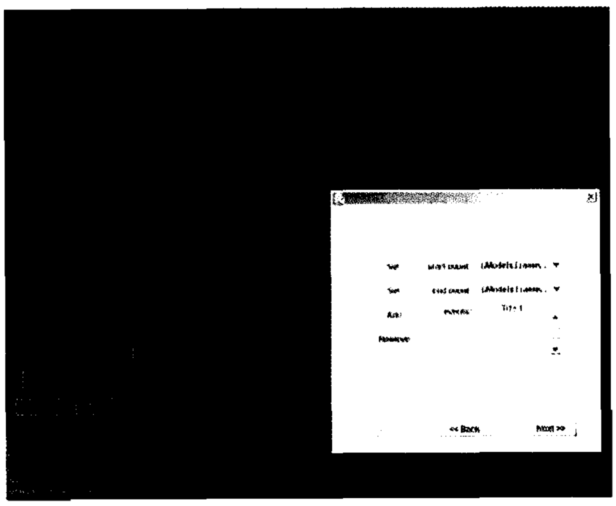

Fig. 9. Analysis and conversion of discrete simulation events.

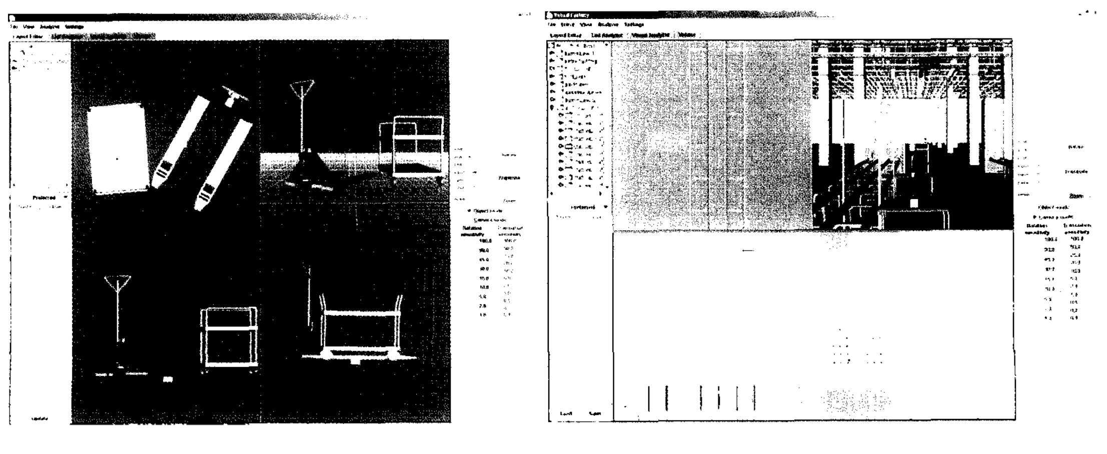

Fig. 10. The shop floor layout.

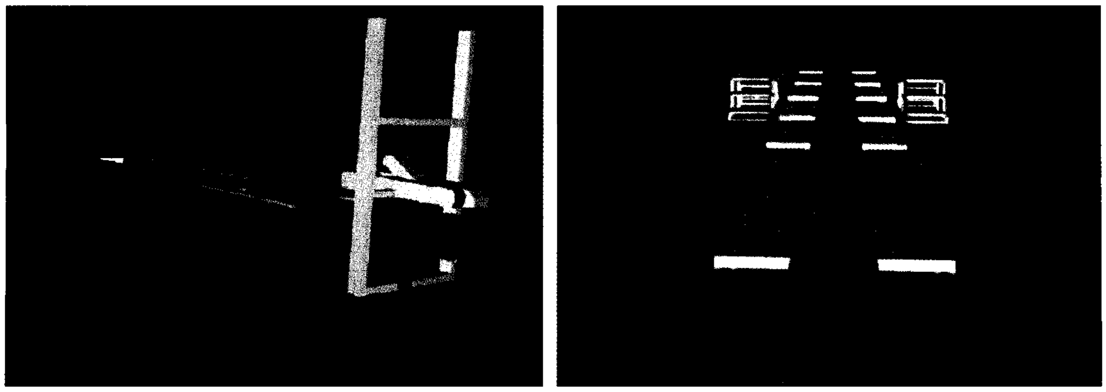
Fig. 11. Testing the behaviors of animated objects.

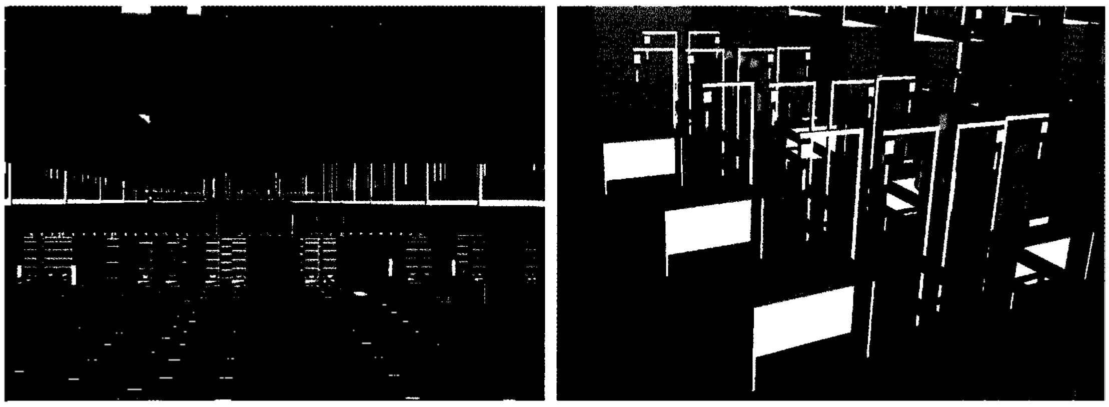
Fig. 12. Virtual shop floor.

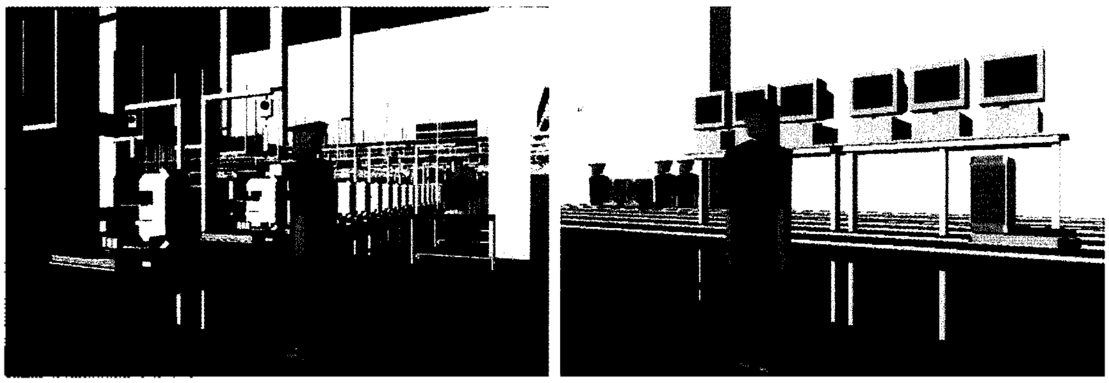
Fig. 13. Virtual assembly and test lines.

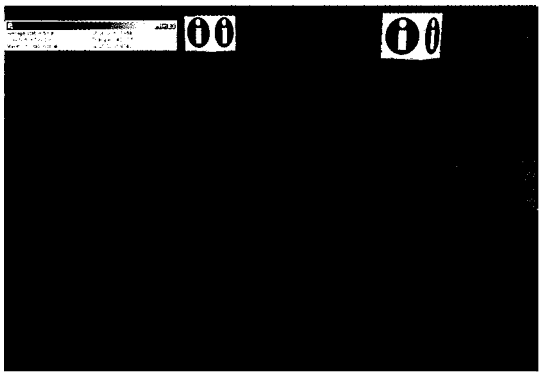
Fig. 14. Virtual shop floor with additional information.

of the shop floor, Fig. 11 shows an example of testing the behaviors of animation objects, Fig. 12 shows the virtual shop floor and Fig. 13 shows the virtual assembly and test lines. There are some types of information that cannot be visualized as animated 3D models. These types of information contain valuable background information about the system parts and dynamic processes of the system. Such information is also provided for users to understand the manufacturing environment and the progress of the manufacturing process, although not appropriate to be included through animation. Fig. 14 shows the virtual shop floor with additional information.

6. Conclusions and Future Work

Presented in this paper is a new approach for 3D visualization of discrete event simulation. By developing new methods to analyze discrete simulation results and to convert them into animation events for the generation of 3D animation, this approach creates an innovative and efficient connection between discrete event simulations and 3D animations. In addition, it develops a 3D animation framework for the visualization of discrete simulation results. This framework uses the software component technology, and supports the reuse of animation objects and behavior components. It allows rapid development of animations from simulation results. By means of a library of 3D animation objects and easy-to-use graphics editors, this visualization system does not require its users to have any special knowledge of computer graphics. The presented approach has been successfully applied in virtual manufacturing for visualizing the discrete simulation results of the entire assembly processes for an electronics assembly factory.

Future research will focus on the synchronous integration of 3D visualization with discrete event simulations. The main work includes generating animation events and the layout for any given scene automatically from simulation logic. A knowledge-based approach will be established to translate the simulation logic of certain simulators automatically into animation events and scene layouts.

Acknowledgements

This work is supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] Banks, J. (1999), Discrete event simulation, Proceedings of the 1999 Winter Simulation Conference, 7-13.

[2] Birta, L.G. and Ozmizrak, F.N. (1996), Knowledge-based approach for the validation of simulation models: the foundation, *ACM Transactions on Modeling and Computer Simulation*, **6**(1), 76-98.

[3] Blanchebarbe, P. and Diehl, S. (2001), A framework for component based model acquisition and presentation using Java 3D, ACM Proceedings of the sixth international conference on 3D Web technology, Paderbon, Germany, 117-125.

[4] Dessouky, M.M., Verma, S., Bailey, D.E. and Rickel, J. (2001), A methodology for developing a web-based factory simulator for manufacturing education, *IIE Transaction*, **33**(3), 167-180.

[5] Elcacho, C., Schafer, Aa, Dorner, R. and Luckas, V. (1998), Performing 3D scene and animation authoring tasks efficiently: an innovative approach, IEEE Proceedings of Computer Graphics International, Hannover, Germany, 242-244.

[6] eM-Plant reference manual (2002), version 4.6, Tecnomatix Tecnologies Ltd.

[7] Fayad, M.E., Schmidt, D. and Johnson, R. (1999), *Building application frameworks: object-oriented foundations of framework design*, New York: Wiley, ISBN 0471248754.

[8] Gennart, B.A. and Luckham, D.C. (1992), Validating discrete event simulations using event pattern mappings, Proceedings of the 29th ACM/IEEE Design Automation Conference, 414-419.

[9] Geuder, D. (1995), Object oriented modeling with Simple++, Proceedings of the 1995 Winter Simulation Conference, 534-540.

[10] Hamilton, G. (July, 1999), The JavaBeans^{TM} 1.01 specification, available online from http://java.sun.com/beans/ docs/spec.html.

[11] Hoeger, H. and Jones, J. (1996), Integrating concurrent and conservative distributed discrete-event simulators, *Simulation for Understanding*, **67**(5), 303-314.

[12] Kamat, V.R. and Martinez, J.C. (2000), 3D visualization of simulated construction operations, Proceedings of the 32nd Winter Simulation Conference, Orlando, USA, Vol. 2, 1933-1937.

[13] Kamat, V.R. and Martinez, J.C. (2001), Enabling smooth and scalable dynamic 3D visualization of discrete-event construction simulations, Proceedings of the 33rd Winter Simulation Conference, Arlington, USA, Vol. 2, 1523-1533.

[14] Kelsick, J.J. and Vance, J.M. (1998), The VR factory: discrete event simulation implemented in a virtual reality environment, 1998 ASME Design Engineering Technical Conferences, Atlanta, Georgia (CD-ROM).

[15] Kheir, N.A. (1996), *System modeling and computer simulation*, New York, Marcel Dekker.

[16] Klingstam, P. and Gullander, P. (1999), Overview of simulation tools for computer-aided production engineering, *Computers in Industry*, **38**(2), 173-186.

[17] Law, A.M. and Kelton, W.D. (2000), *Simulation Modeling and Analysis*, 3rd Edition, The McGraw-Hill Companies, Inc.

[18] Luckas, V. and Broll, T. (1997), CASUS: an object-oriented three-dimensional animation system for event-oriented simulators, Proceedings of Computer Animation, Geneva, Switzerland, 144-150.

[19] Luckas, V. and Dörner, R. (2000), Experiences from the future - using object-oriented concepts for 3D visualization and validation of Industrial Scenarios, *ACM Computing Surveys*, **32**(1).

[20] Mueller-Wittig, W., Jegathese, R. and Song, M., etc. (2002), Virtual Factory-Highly Interactive Visualisation for Manufacturing, Proceedings of the 2002 Winter Simulation Conference, San Diego, California, Vol. 2, 1061-1064.

[21] Nierstrasz, O., Gibbs, S. and Tsichritzis, D. (1992), Component-oriented software development, *Communications of the ACM*, **35**(9), 160-165.

[22] Praehofer, H., Sametinger, J. and Stritzinger, A. (1999), Discrete Event Simulation using the JavaBeans Component Model, International Conference On Web-Based Modelling & Simulation, San Francisco, California, also available online from http://www.swe.uni-linz.ac.at/publications/.

[23] Quest (2000), Delmia Corporation, http://www.delmia.com/.

[24] Quick, J.M. (2002), Component-based 3D Visualization of Simulation Results, The Advanced Simulation Technologies Conference, San Diego, USA, also available online from http://www.scs.org/scsarchive.

[25] Quick, J.M. (2003), Monitoring and Control of Systems by Interactive Virtual Environments, 10th International

Conference on Human-Computer Interaction, Crete, Greece, 1101-1105.

[26] Rohrer, M.W. (2000), Seeing is believing: The importance of visualization in manufacturing simulation, Proceedings of the 2000 Winter Simulation Conference, 1211-1216.

[27] Rohrer, M.W. and McGregor, I.W. (2002), Simulating Reality Using AutoMod, Proceedings of the 2002 Winter Simulation Conference, Salt Lake City, USA, 173-181.

[28] Salisbury, C.F., Farr, S.D. and Moore, J.A. (1999), Web-based simulation visualization using JAVA3D, Proceedings of the 1999 Winter Simulation Conference, Vol. 2, 1425-1429.

[29] Sametinger, J. (1997), *Software engineering with reusable components*, Springer-Verlag, ISBN 3-540-62695-6.

[30] Shukla, C., Vazquez, M. amd Chen, F.F. (1996), Virtual manufacturing: an overview, *Computers & Industrial Engineering*, **31**(1-2), 79-82.

[31] Sowizral, H., Rushforth, K. and Deering, M. (1999), Java 3D™ API Specification, available online from http://java.sun.com/docs/books/java3d/.

[32] Sun, L. and Ning, R. (2002), The research on constructing of the virtual shop-floor environment, The Third International Conference on Virtual Reality and its Application in Industry, Hangzhou, China, 389-394.

[33] Wenzel, S. and Jessen, U. (2001), The integration of 3-D Visualization into the simulation-based planning process of logistics systems, *Simulation*, **77**(3-4), 114-127.

[34] Witness VR (2003), Lanner Group, http://www.lanner.com/

[35] Zhai, W., Fan, X., Yan, J. and Zhu, P. (2002), An integrated simulation method to support virtual factory engineering, *International Journal of CAD/CAM*, **2**(2), 39-44.

[36] Zhou, H., Tan, H.S. and Sivakumar, A.I. (2001), Digital Models for Manufacturing Process Visualization, Proceedings of International Conference on Integrated Logistics, Singapore, 113-122.

Yongmin Zhong is a research fellow at Department of Mechanical Engineering, Monash University, Australia. His research interests include virtual reality, geometric modelling and CAD/CAM. Prior to joining Monash University, he worked as a postdoctoral fellow at School of Computer Science, University of Windor, ON, Canada. He also worked as a research fellow at School of Computer Engineering, Nanyang Technological University, Singapore and a lecturer at Department of Aeronautical Manufacturing, Northwestern Polytechnical University, China. He obtained a BSc degree in 1990 from Northwest University (China), an MSc degree in 1993 and a Ph.D. degree in 2000 from Northwestern Polytechnical University (China).

Xiaobu Yuan is an Associate Professor with the School of Computer Science, University of Windsor, ON, Canada. His research interests include advanced human/computer interaction, robotics, intelligent systems, and software engineering and testing. He received the B.Sc. degree in computer science from the University of Science and Technology of China, Hefei, in 1982, the M.S. degree in computer science from the Institute of Computing Technology, Academia Sinica, Beijing, China in 1984, and the Ph.D. degree also in computer science from University of Alberta, Edmonton, AB, Canada, in 1993. Dr. Yuan has been a reviewer of international journals and a member in program committees of international conferences, including IEEE ICRA'2000, ICRA'2002, and ICRA'2004.

Yongmin Zhong

Xiaobu Yuan