Dynamic Three-Dimensional Visualization of Fluid Construction Materials

Vineet R. Kamat, S.M.ASCE,[1] and Julio C. Martinez, M.ASCE[2]

Abstract: The presented work extends the state-of-the-art of visualizing discrete-event construction simulations in three dimensions (3D). Efficient methods are presented along with a tool, *ParticleWorks*, that can be used to animate simulated construction processes that involve unstructured, fluid construction materials as resources or byproducts. Common construction processes that involve such fluid materials include placing concrete, dumping dirt, shotcreting, sandblasting, dewatering, water distribution, and inserting slurry. The writers capitalize on a classical computer graphics concept called particle systems to design simple, simulation model-authorable, parametric-text methods that can describe arbitrary volumes of dynamic fluid construction materials in animated 3D virtual construction worlds. These methods can be used to instrument discrete-event simulation models (or other external authoring interfaces) to automatically generate dynamic visualizations of any modeled construction operations that commonly handle and process fluid construction materials.

DOI: 10.1061/(ASCE)0887-3801(2004)18:3(237)

CE Database subject headings: Simulation; Construction materials; Computer graphics; Particles; Fluid dynamics.

Introduction

In discrete-event simulation (DES) analyses, the ability to see a three-dimensional (3D) animation of an operation that has been modeled and simulated allows for three very important things: (1) the developer of the simulation model can confirm that there are no errors in the coding (verification); (2) the domain experts, field personnel, and decision makers can discover differences between the way they understand the operation and the way the model developer understands it (validation); and (3) the model can be communicated effectively offering decision makers valuable insight into operational details that would otherwise be nonquantifiable and nonpresentable. This, coupled with verification and validation, makes simulation models credible and encourages their use in making decisions (Law and Kelton 2000; Kamat and Martinez 2003a).

The current state-of-the-art in 3D visualization of simulated construction processes allows the animation of a limited number of construction processes. These include processes in which resources (equipment and crew) always move on fixed well-defined paths, processes that do not require any physical deformation of rigid resources, and in particular processes that do not involve use of unstructured, fluid construction materials (Kamat and Martinez 2001). Numerous unstructured materials generally capable of flowing are, however, commonly handled and processed on typical construction sites. Indeed, materials such as concrete, dirt, mortar, sand, slurry, and water are together central to most construction operations. Common simulated construction processes such as dumping dirt, distributing water, dewatering caissons, placing concrete, sandblasting, shotcreting, and slurry-wall construction cannot be animated in smooth, continuous 3D worlds unless methods to accurately represent dynamic volumes of the involved fluid construction materials are designed.

Main Contribution

The research this paper presents explores several original concepts to design effective animation methods to describe the dynamics of unstructured, fluid construction materials in 3D animations of discrete-event construction simulation models. Using principles of Newtonian mechanics where appropriate, the writers designed techniques that allow the accurate motion, changes of form, and dynamics of fluid material volumes to be expressed using simple, simulation model-authorable, parametric statements of text. The work thus puts in place the technology that allows the 3D animation of any simulated construction processes that involve common, unstructured, fluid construction materials as resources or by-products.

Background

In construction, different people understand different things by the term ''visualization.'' In particular, the term has been used in the literature to refer to any kind of series of sequential computer frames without taking into account their origin or their contents (Op den Bosch 1994). In effect, numerous computer-based visual activities that can be directly or indirectly used in construction planning may be appropriately termed visualization. These activities include, but are not limited to, (1) manual coding and/or

[1]Assistant Professor, Construction Engineering and Management Program, Dept. of Civil and Environmental Engineering, Univ. of Michigan, Ann Arbor, MI 48109. E-mail: vkamat@engin.umich.edu

[2]Associate Professor, Dept. of Civil and Environmental Engineering, Virginia Tech, 200 Patton Hall, Blacksburg, VA 24061-0105. E-mail: julio@vt.edu

Note. Discussion open until December 1, 2004. Separate discussions must be submitted for individual papers. To extend the closing date by one month, a written request must be filed with the ASCE Managing Editor. The manuscript for this paper was submitted for review and possible publication on October 18, 2002; approved on August 14, 2003. This paper is part of the ***Journal of Computing in Civil Engineering***, Vol. 18, No. 3, July 1, 2004. ©ASCE, ISSN 0887-3801/2004/3-237–247/$18.00.

interactive creation of specific virtual construction scenarios using graphics libraries (e.g., World Tool Kit) or interactive tools (e.g., Bentley Dynamic Animator) (e.g., Tsay et al. 1996); (2) 3D CAD model-based animation of CPM construction schedules (i.e., 4D CAD) (e.g., Koo and Fischer 2000); (3) visualization of assembly sequences and real-time virtual interactive modeling of construction equipment (e.g., Interactive Visualizer ++) (e.g., Op den Bosch 1994); (4) virtual reality (VR) construction equipment simulators (e.g., Park 2002); and (5) CAD model-based project information access over the internet using VRML (e.g., Lipman and Reed 2000).

The context of this research differs from those listed above and is about enabling accurate, smooth, continuous, and dynamic 3D animation of construction processes modeled using DES tools (Kamat and Martinez 2001, 2002). The presented work is specifically concerned with extending the state-of-the-art of 3D visualization of simulated construction processes by designing animation methods that can describe and depict dynamic volumes of fluid construction materials used as resources or generated as byproducts in common construction processes.

Practical Motivation

The measure of virtual realism necessary or sufficient to fully exploit 3D visualization of simulated construction processes is often a matter of subjective opinion. While simple, symbolic, iconic 2D animation is adequate for some, others (including the writers) strive for detailed 3D visualization in immersive, photorealistic virtual worlds. Notwithstanding the scientific community's polarization, it is undeniable that in general, engineers' expectations from computer graphics have increased greatly in recent years. This can be largely attributed to the impressive computer-generated sequences the motion picture industry creates in animated films and special effects.

What these expectations signify for 3D animation of simulated processes is that displayed graphics must be faithful, comprehensive, and should not need any verbal or textual clarification (Rohrer 2000). Since visualization is principally a communication tool, engineers must be able to animate faithful, realistic depictions of all modeled real-world processes (Farr and Sisti 1994). Construction engineers, for instance, must be able to graphically see processes such as concrete being placed into virtual forms, dirt pouring out from virtual dump trucks, and mortar being projected at high velocity from virtual shotcrete applicators. Without them, it is impossible to accurately animate and communicate most discrete-event construction process models in 3D.

Theoretical Motivation

Volumes of fluid construction materials such as concrete, dirt, gravel, mortar, sand, slurry, and water naturally do not have fixed or deterministic shapes and forms. Such materials flow under the influence of prevailing natural (e.g., gravity) and imparted (e.g., pump pressure) forces until physical equilibrium is established. Computational fluid dynamics literature provides classic models such as the Navier-Stokes equations to describe the motion and behavior of flowing liquids. Such models, although highly accurate, require very intensive computation to solve. The dynamics of rigid bodies, on the other hand, can be described using classical Newtonian mechanics.

Methods designed to allow DES models to automatically describe processes involving fluid construction materials demand that the methods themselves be simple. Simplicity, in this context, means that the syntax of the methods and the values sought by their parameters are both within the authoring capabilities of external processes such as DES tools. In addition, the methods must incorporate rich semantics so that the necessary descriptions are communicated accurately with minimal interprocess interaction.

Regular objects in virtual environments are generally represented using surface-based, polygonal CAD models. Such objects typically have fixed, well-defined, smooth surfaces. Fluid object surfaces, on the other hand, are irregular, complex, and ill-defined and cannot generally be represented with smooth surfaces inside 3D virtual worlds.

The issues that motivated this work were thus threefold. First, feasible procedures to accurately describe the motion and dynamics of arbitrary volumes of fluid construction materials had to be investigated (i.e., the physical model). Second, simple methods to encapsulate the physical descriptions in a finite number of parametric-text statements needed to be designed (i.e., the interface model). Finally, techniques to represent the descriptions in animated 3D virtual construction worlds had to be explored such that the results look visually convincing, move realistically, and can be animated in real time (i.e., the implementation model). These issues were tightly intertwined and thus had to be concurrently addressed.

Initiative

In attempting to address these issues, the writers found most promise in a technology based on the concept of the particle system. Particle systems represent a classical technique of describing unstructured fluid objects in interactive 3D virtual environments. In addition, the semantics of the technique are such that the approach can incorporate any physical computational model to describe the appearance or dynamics of fluid objects. As the following discussion will clarify, this is a significant feature that facilitates immensely the design of generic methods to describe dynamic fluid construction materials in animated 3D worlds.

Reeves (1983) coined the term ''Particle system.'' The term was used to describe a method that uses an arbitrary number of geometric primitives (called particles) in virtual space to visually represent certain objects or effects that cannot be represented using conventional techniques. An object represented by such a particle system was called a ''fuzzy object.'' Examples of physical phenomena and materials that can be called fuzzy objects include clouds, smoke, water, and fire. Reeves and his colleagues first applied this concept in creating the wall of fire element from the Genesis Demo sequence of the film ''*Star Trek II: The Wrath of Khan*'' (Paramount 1982).

Their astonishing results have since pioneered several research works on realistic representations of fuzzy objects in virtual worlds. Reeves' specification of a ''particle'' and how particle systems are generally computed prevails through most subsequent related research (e.g., Dorsey et al. 1996; Foster and Metaxas 2000; Carlson et al. 2002). Fuzzy objects by definition do not have smooth and well-defined surfaces. On the contrary, their surfaces are irregular, complex, and ill-defined. Describing realistic visual representations of such fuzzy, fluid objects in virtual environments generally proves difficult with conventional polygonal, surface-based techniques of computer-image synthesis (Reeves 1983; Foster and Fedkiv 2001).

Fluid construction materials such as dirt, sand, mortar, concrete, and slurry may be appropriately termed fuzzy as they are not rigid objects and are typically capable of flowing until a stable physical equilibrium is established. The dynamics of such objects

(e.g., concrete flowing from a bucket, dirt being emptied from a truck, etc.) cannot be visually represented using simple geometric transformations commonly used in computer graphics. The concept of the particle system thus presents an ingenious technique to describe the dynamic and fluid changes in the shape and appearance of such material masses in animations of simulated construction processes.

Particle Systems

A particle system is an abstraction for simultaneously operating on many similar objects that all move and change according to the same basic rules, no matter what the objects and rules are. A particle system is realized as a simple collection of many particles that together represent a fuzzy object. The important issue when describing a particle system is the overall appearance of the entire object it represents. Reeves (1983) describes the following three basic ways in which particle system representations differ from other techniques of describing physical objects: (1) an object is represented as a cloud of primitive particles that define its volume, and not by a set of primitive surface elements, such as polygons or patches, that define its boundary; (2) a particle system is not a static entity. Its particles change form and move with the passage of time; new particles are ''born'' and old particles ''die''; and (3) an object represented by a particle system is not deterministic, since its shape and form are not completely specified. Instead, stochastic processes specify parameters that are used to create and change an object's shape, appearance, and behavior.

Particle Attributes

Each particle contained in a particle system can be thought of as an entity with a set of attributes that ultimately dictate the particle's behavior and appearance. A particle's attributes typically include mass, color, velocity, size, and age. A particle in this context is therefore not necessarily a very tiny speck but can represent many different things. Each particle within a particle system may be represented as a geometric primitive (e.g., point, line, triangle, quadrilateral, etc.), a water droplet, a grain of dirt, or even a complex multipolygon CAD model (e.g., bird, human, blade of grass, etc.).

Recursive Model

Fig. 1 schematically presents the sequence of actions that must be performed to describe each frame in a motion sequence involving particle-system–represented fuzzy objects. The following minimal steps are performed:

- New particles are generated into the system; each new particle is assigned its individual attributes by sampling from the specified ranges of initial conditions; any particles that have existed within the system past their prescribed lifetime are destroyed.
- An image of the remaining particles is rendered, i.e., drawn on the screen.
- The future course of the particles is calculated according to their dynamic attributes; active forces acting on each particle are accumulated based on a particle's physical characteristics and its current physical state.
- The particle attributes are transformed to new values by integrating over the elapsed time interval.

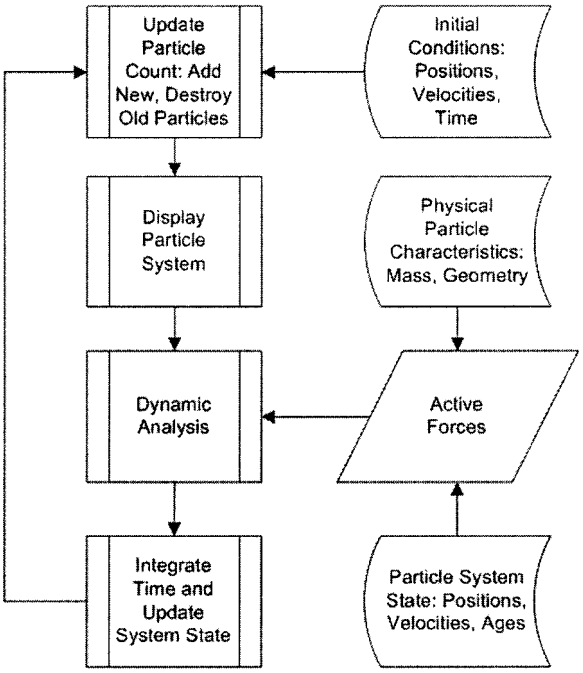

Fig. 1. Basic particle system model

Particle Dynamics

Individual particles within a particle system move in three-dimensional space and may also change other attributes such as color and size over time to depict the evolution of the fuzzy object they represent. At each instant in time, particles can be moved from one position to the next by simply knowing their current position and current velocity (magnitude and direction). The rules that govern how a particle system behaves and evolves over time may be specified and interpreted as a set of arbitrary forces that act on each particle within the system. A particle system typically uses an acceleration factor to modify the velocity of its particles at each instant. This attribute can be used to simulate external physical forces (e.g., gravity) that might cause particles to move in parabolic arcs rather than in straight lines, for instance. The evolution of the particle system over time, i.e., the shape and form of the represented fuzzy object, is then the result of all specified forces acting on each constituent particle.

The forces acting on the particles at each instant can be resolved into their corresponding accelerations using Newton's fundamental laws of mechanics. The accelerations themselves represent the rates of change in the velocities of the particles. Applying fundamental results from calculus and using numerical integration techniques, it is possible to obtain the new velocity and subsequently the new position of each particle in the particle system. In addition to specifying the forces that act on the system, it is also necessary to specify rules that are responsible for destroying particles in the system and emitting new ones as needed. Over a period of simulated time, particles are thus generated into a system, move and change form within the system, and die from the system.

Technical Approach

Particle systems can be used to visually represent phenomena that do not necessarily conform to the physical laws of nature humans

experience and understand. The ''forces'' each particle in a particle system is subjected to inside a virtual world need not necessarily have physical world counterparts. As a corollary, particles need not be subjected to even the obvious physical world forces (e.g., gravity). Particle systems can thus be used in several ways to represent interesting visual effects that might generally not occur in the real physical world. The motion picture industry effectively exploits this result to create dramatic special effects in feature films.

Construction operations, however, are performed in the real world. In performing construction, the crew, equipment, and materials all obey the generally understood physical laws of nature. For instance, the fact that concrete from a bucket flows into a form when the latch is released can be attributed to the Earth's gravitational force. Gravity acts on each ''globule'' of fresh concrete, forcing it out of the bucket and downward towards the form. Similarly, consider the process of pumping water out of a caisson. The parabolic path the gushing water follows before striking a surface is a function of both gravity and the pressure with which the water is pumped out. All construction materials (fluid or otherwise) follow the rules of natural physics. Examples of physical phenomena (in addition to gravity) that can influence the behavior of common materials include viscous drag (i.e., air resistance), friction, and resilience. Particle systems that represent fluid construction material masses must therefore evolve under the influence of such real-world force counterparts.

Pertinent Mechanics

In order to visually describe fluid construction materials, each particle in a particle system is represented as a point mass acted upon by the prevalent natural and imparted physical forces. Included in Newton's results is the recognition that mass and velocity of the particles are what affect them (and a system). In particular, the sizes and shapes of the particles can be disregarded and they can be treated as point masses. To describe the particle system's dynamic behavior, the resultant of the forces acting on each particle is then resolved. The mechanics of this method hinge on the fundamental relationship between the resultant force (f) imparted to each particle, the mass of that particle (m), and the acceleration (a) of that particle. This can be stated in the familiar Newtonian notation as $f=ma$. Assuming that particle masses are fixed, equations that express how position and velocity change over time can be derived from this law.

The velocity (v) of a particle is simply the first derivative with respect to time of its position (x), i.e., $v=dx/dt=\dot{x}$. A particle's acceleration is similarly the first derivative with respect to time of its velocity (v) and the second derivative with respect to time of its position (x), i.e., $a=f/m=dv/dt=\dot{v}=d^2x/dt^2=\ddot{x}$. In this context, given a set of forces acting on several particles at any time instant t, the intention is to determine the location of each of the particles at the next time instant $t+h$ (i.e., after a small amount of time h has passed). Given all active forces and the masses of the particles, the acceleration of the particles can be obtained. That acceleration can then be integrated with respect to time (t) to determine the new velocity of each particle. By integrating again, each particle's new position can be calculated.

Particle acceleration (a) is the second derivative with respect to time of its position (x). Introduction of the first derivative quantity [velocity (v)] allows the consideration of only a system of first-order ordinary differential equations (ODEs) that each determines a function of time. Describing the evolution of particles in a dynamic particle system is thus an initial value problem. Given an initial state (position and velocity) and active forces, the dynamic behavior of each particle in a system can be evolved over time.

Computation Scheme

In this work, the described initial value problem is solved by numerical approximation. Several methods suitable for solving such a class of problems exist in the mathematical literature. Some classical single-step numerical integration methods include Euler's method, the Midpoint method, and the higher-order Runge-Kutta methods. These methods generally differ in the amount of computation required to arrive at an approximation, the accuracy of the computed solution (i.e., the approximation error), and the stability of the computed solution. In general, higher accuracy and stability come at a corresponding computation cost. Additional discussion of these and other classes of numerical integration methods from the computation perspective can be found in Press et al. (1992).

Few numerical integration methods are, however, suitable for real-time particle system animation on the computer. Many methods adapt the integration step size to the function itself, which in this case is the path of each particle. However, giving each particle its own step size requires much more computation. In addition, this would also cause each attribute of a particle to require a different, conflicting step size. Euler's forward integration method is used in this research. This method is not considered to be exceptionally accurate or stable in the mathematical literature. However, a survey of literature describing applications of the particle system concept suggests that this method is popular due to its simplicity and computational efficiency (McAllister 2001; Witkin and Baraff 2001). In addition, the method allows the accuracy of the computed solutions to be interactively adjusted by varying the step size of integration. The quality of the achieved visualizations can thus be a function of the available computing power. The writers were thus naturally inclined to adopt this method in the presented work.

Euler's forward integration method simply approximates the next value of the function x of t by the sum of the first two terms of the Taylor series expansion. Mathematically, it states that for sufficiently small values of the integration step size h, $x_{n+1}\approx x_n+h(dx/dt)_n$. The truncation error in x_{n+1} depends on the step size h. By selecting h to be of sufficiently small size, the difference between the real function value and the approximation can be forced to be less than some required error magnitude e (Burden and Faires 1989). However, there is a price to be paid. As the step size h is decreased, the number of steps and computation required to bridge an interval increases.

Iterative Algorithm

A discrete-time approximation is used when applying forces to particles. This implies that forces are applied to the particles at a particular instant in time as if the forces' effect accumulated over a small time interval h. The simulation clock is then advanced by the length of the interval and the forces are then reapplied with the particles having their updated values. Only the unary forces of gravity and viscous drag are considered in the analysis. These forces act independently on each particle, either exerting a constant force or one that depends on particle velocity. Gravity is a constant force that is applied to all particles. The viscous drag (i.e., air resistance) that each particle is subjected to is taken to be a function of the particle's velocity.

Consider a mass of concrete flowing into a form from a bucket when the latch is released. For purposes of visual simulation only, the flowing mass is considered to be a stream of concrete particles. Each concrete particle is then treated as an object falling under the influence of gravity. For simplicity, the forces of spatial interaction such as adhesion, surface tension, and attraction that may act on any or all pairs of concrete particles are ignored. All dynamic particle attributes are represented as three-dimensional vectors that encapsulate both magnitude and direction where appropriate. For instance, particle velocity is interpreted as a vector having its tail at the particle's position and its tip at a particular point inside 3D Euclidean space. The vector's length and direction together specify both the magnitude and the direction of a particle's velocity. Any force acting on a particle and the resulting acceleration are similarly interpreted as directional vectors. A particle's position, on the other hand, is interpreted as a three-dimensional vector that specifies only its location in 3D space. In the subsequent discussion, all attributes interpreted as three-dimensional directional vectors are denoted with an arrowhead above the attribute's symbol. All the ideas of integral curves, numerical approximations, etc. carry over intact to three-dimensional space. The only change is in the interpretation of the resulting trajectories.

The force of gravity is represented by the familiar force law $\vec{f}_g = m\vec{G}$, where the constant acceleration $\vec{G}$ is 9.81 m/s^2 acting downward towards the Earth's surface. Viscous drag or air resistance is taken to be proportional to velocity and is represented as $\vec{f}_{drag} = -k\vec{v}$, where the empirical constant k is the coefficient of drag. The direction of the drag force is against the velocity direction. Based on the earlier discussion, the following straightforward equation can now be expressed:

$$\frac{d\vec{v}}{dt} = \frac{m\vec{G} - k\vec{v}}{m} = \vec{G} - \left(\frac{k}{m}\right)\vec{v} \tag{1}$$

Using an Euler solver, the value of the velocity at the end of time step $n+1$ is now given by $\vec{v}_{n+1} \approx \vec{v}_n + h\vec{a}_n$, where $\vec{a}_n$ is the net acceleration as given by Eq. (1). If air resistance is neglected, the net acceleration $\vec{a}_n$ is replaced by the constant $\vec{G}$. Otherwise, the algorithm proceeds exactly as in the case of the nonuniform acceleration considered above. This is also true when forces other than gravity and drag are active. All the active force vectors are simply accumulated in formulating Eq. (1). A similar procedure is used to formulate the equation for the value of each particle's position at time step $n+1$. The other first-order differential equation is used to get $x_{n+1} \approx x_n + h\vec{v}_n$. Thus, $dx/dt = \vec{v}$ is integrated at the same time as $d\vec{v}/dt = \vec{a}$ in an alternating manner. The two coupled ODEs thus provide an iterative algorithm for computing particle positions resulting from any arbitrary resultant force on the instantiated particle systems.

To implement this algorithm, then, the initial position (x_0) and the initial velocity $\vec{v}_0$ of each particle in the system must be specified. The velocity $\vec{v}_0$ at time t_0 can then be used to calculate the value of $\vec{a}_0$, the net acceleration at time t_0. The values, along with x_0, can then be used to calculate the position and velocity at time $t_1 = t_0 + h$. These values along with the net acceleration at time t_1 are then used to calculate the values at t_2 and so on.

ParticleWorks

The writers' implementation of particle systems and the algorithms that manipulate them is a tool that allows accurate visualization of masses of fluid, unstructured construction materials in smooth, continuous, animated 3D virtual worlds. This tool, called *ParticleWorks*, is implemented as an extension (add-on) to the VITASCOPE visualization system. VITASCOPE is a user-extensible 3D animation language designed specifically for visualizing simulated construction operations in smooth, continuous, animated 3D worlds (Kamat and Martinez 2003b). A limited subset of the VITASCOPE language and the corresponding prototype implementation were referred to as the Dynamic Construction Visualizer (DCV) in prior publications (Kamat and Martinez 2001, 2002). VITASCOPE is the tangible outcome of the writers visualization research efforts that focus on designing automated, process simulation-driven methods to visualize construction processes and products in dynamic 3D virtual worlds.

The *ParticleWorks* add-on extends the VITASCOPE animation language. The add-on defines animation language statements that allow instantiation and interactive manipulation of dynamic particle systems. The parametric-text statements can be used together in interesting ways to describe animated, realistic-looking masses of common fluid construction materials such as dirt, concrete, slurry, and water in visualizations of simulated processes. This is achieved by instrumenting (i.e., programming) discrete-event simulation models to automatically generate a dynamic animation trace during a simulation run (Kamat and Martinez 2001).

ParticleWorks provides statements that allow users to specify the static (e.g., mass, color, size) and dynamic (e.g., initial position, initial velocity) properties of the particles in an instantiated particle system. In addition, the add-on implements several statements that help define properties that influence the behavior of an instantiated system as a whole. Examples of such properties include surfaces in the virtual world off which particles bounce rather than sink in (i.e., ''die''), surfaces which confine system particles rather than permit leakage/penetration (e.g., concrete forms), the number of particles to generate per time instant, and the lifetime of constituent particles (i.e., the number of simulation time units after which created particles must ''die'').

Table 1 presents the major statements *ParticleWorks* implements and briefly indicates their usage. The meaning and significance of these statements will be clear from the examples presented ahead. In general, *ParticleWorks* statements are quite readable. In addition, individually describing every *ParticleWorks* statement and providing detailed instructions on its usage is beyond the scope of this scientific paper. Information of that nature is described in another relevant document (Kamat and Martinez 2003c). Similar detailed information on the usage of the core VITASCOPE animation language is presented in (Kamat and Martinez 2003d).

Fluid Object Definition

ParticleWorks represents each instantiated fluid object as a dynamic particle system. The fluid object is visible only after the source of the particle system that represents that object emits visible particles. For instance, consider a water sprinkler truck that distributes water over haul roads on earthmoving jobsites. Water distribution is commonly used in earthmoving to increase the moisture content of the soil to the optimum range for compaction, and to reduce dust along haul routes (Peurifoy and Schexnayder 2002). Distributor trucks drive over haul routes spraying water (at varying rates) from linearly arranged nozzles on rear-mounted spray bars.

In order to animate this operation, the spraying water (the fluid object) can be described as a stream of particles emitted from a linear source particle system that is attached to a normal polygo-

Table 1. Usage of Selected *Particle Works* Statements

Statement	Usage
FUZZYOBJECT [ObjName] [...];	Instantiate and initialize a particle system source
[ObjName].SetVelocity [DomainType] [Param];	Set the initial velocity of instantiated particles
[ObjName].SetSource [DomainType] [Param];	Set the source where particles are generated, if not set with FUZZYOBJECT
[ObjName].SetRate [ParticlesPerUnitTime];	Set the number of particles entering the system at each time instant
[ObjName].SetAge [AgeLimit];	Set how long particles stay in the system before being destroyed
[ObjName].SetSink [SinkLevel];	Set the elevation level below which entering particles are destroyed
[ObjName].SetColor [DomainType] [Param];	Set the color of instantiated particles
[ObjName].SetMass [Value];	Set the mass of instantiated particles
[ObjName].SetPrimitive [PrimitiveType];	Set what geometric primitive is used to draw particles (points or lines)
[ObjName].AddObstruction [DomainType] [Param];	Add a physical barrier surface obstructing particle flow (e.g., concrete forms)
[ObjName].SetGravity [NewGValue];	Change values of $\vec{G}$ to simulate buoyancy underwater or loss of gravity in outer space

nal CAD model of a sprinkler truck. By linear source, it is meant that particles in this system are stochastically generated along a straight line. In this example, the line corresponds to the arrangement of nozzles at the back of the sprinkler truck. The initial positions of the generated particles are thus random points sampled along this straight line.

The initial velocities of the generated particles in this case are a function of the pressure (i.e., force) with which the truck sprays water. The number of particles to generate per time instant is similarly a function of the truck's discharge rate. The particles are destroyed when they strike the virtual ground surface. This corresponds to the absorption of real sprayed water by the soil upon contact. Fig. 2 presents an annotated VITASCOPE animation trace segment that describes this operation. *ParticleWorks*-defined statements as well as core VITASCOPE language commands are used in the description. The significance of each numerical argument used in the statements is discussed in detail in the following subsections.

By introducing a *ParticleWorks* fluid object in a scene (either by attachment to another scene entity or by global placement), the source of particles in that system and the rules that govern their dynamics are specified. It is only after the source begins to generate and emit particles as time passes that the actual volume of the fluid material (spraying water in this example) can be seen. The emitted particles are then continuously subjected to physical analysis to describe their dynamic, real-world-like behavior. The particles are destroyed either when they travel below a specified elevation (i.e., sink plane) such as the ground level of the virtual terrain, or when they have outlived their specified age (measured in simulation time units). Fig. 3 presents a snapshot of the animation achieved by processing the above trace segment in the VITASCOPE virtual environment application.

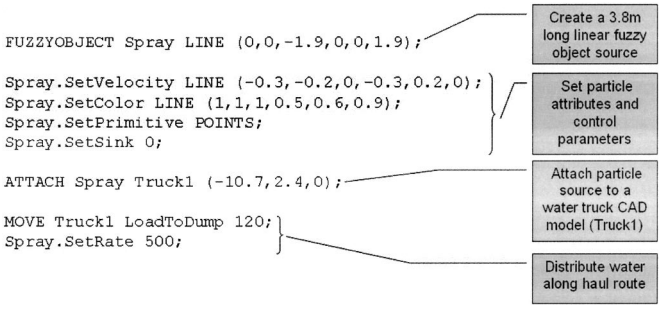

Fig. 2. Defining a water distribution truck

In a real operation, the spraying water travels out of the truck's storage tank to the nozzles through pipes when the valves are opened. However, in an animated virtual world, water droplets (i.e., particles that look like water droplets) are generated just at the point where they are visible (i.e., at the linearly arranged nozzles). The particles are similarly destroyed just at the point where they will no longer be visible. In this case, that occurs when the particles travel below the ground level (represented by a zero elevation value) of the virtual terrain surface on which the water truck travels.

Similarly, in order to describe a mass of concrete pouring out from a bucket, all that must be done is attach a circular disk-shaped particle source to the underside of the concrete bucket model. When the latch is released, the source must emit particles that together resemble a mass of flowing concrete. Fig. 4 presents a trace file segment that describes such a process. A snapshot of the animation achieved by processing this segment is then presented in Fig. 5. Such processes cannot be visualized using only traditional surface-based 3D CAD models. Particle systems make all the difference.

Fluid Object Control

In order to create a dynamic group of particles that exist and move at various times, *ParticleWorks* must initiate particles at the desired place at the desired time. In addition, the particles must exhibit a desired appearance and dynamic behavior. In order to achieve this, *ParticleWorks* must know the rules to be used for selecting initial values of particle properties (e.g., mass, source or

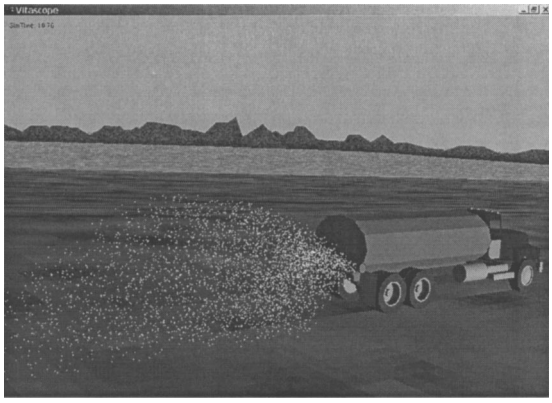

Fig. 3. Animation snapshot of water distribution process

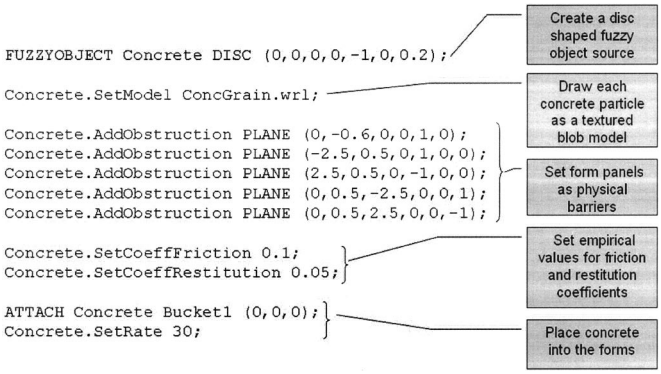

Fig. 4. Describing concrete placement processes

initial position, initial velocity, age). *ParticleWorks* must also know the number of particles to generate each time step and the laws that govern the spatial aspects of their existence. With these data available, *ParticleWorks* can achieve a continuous stream of the desired particle flow.

As Fig. 1 suggests, a particle system can be instructed to perform any set of instructions at each time step. Because this is procedural, this approach can incorporate any computational model to describe the appearance or dynamics of the fluid object. In this research, the writers use (1) bounded stochastic processes to control the shape, appearance, and initial conditions (e.g., position, velocity) of instantiated particles, and (2) physical analysis to control subsequent particle dynamics. In addition, any constant properties and physical constraints are deterministically specified. *ParticleWorks* statements in effect provide access to particle and particle system parameters that control these operations. For instance, in cases where stochastic processes randomly select particle attributes, the selected values are constrained by user-specified parameters. In general, each such parameter specifies a range in which a particle's value must lie.

Concept of Domains

ParticleWorks represents a range with an entity that literature refers to as a Domain (McAllister 2001). A domain can be conceived to be a region of space. For instance, the particle source of an instantiated fluid object is specified and interpreted as a domain. In the examples described above (water truck and concrete bucket), the particle source is specified as a straight line (water truck nozzle arrangement) and a circular disk (shape of the bucket opening), respectively. The source domain describes the region in

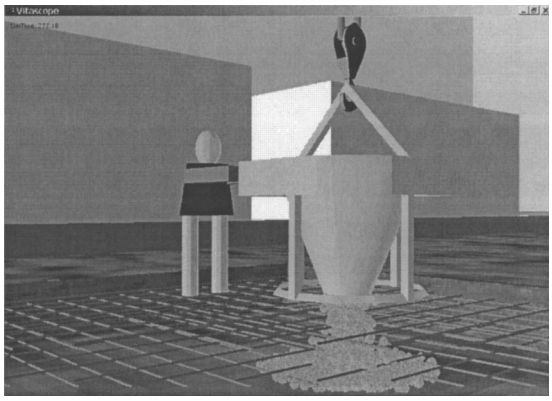

Fig. 5. Animation snapshot of concrete placement process

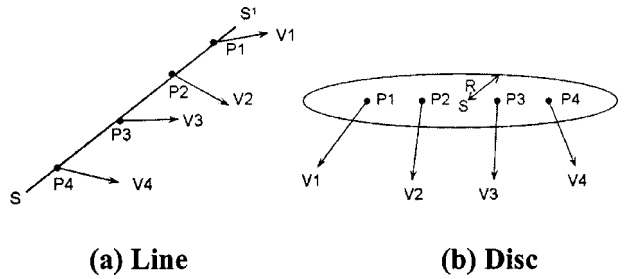

Fig. 6. Source domains

which new particles in a system will be generated. A random point within the domain is then chosen as the initial position of a new particle. Figs. 6(a and b) present this concept graphically for the line and disk domains, respectively. The dots in the domain space indicate randomly chosen positions within each domain. The arrows indicate the magnitude and direction of the initial particle velocities.

In addition to lines and disks, *ParticleWorks* supports points, triangles, planes, rectangles, cylinders, spheres, cones, cubes, and blobs as valid domain spaces. Domains may thus define one-, two-, or three-dimensional regions in space. Table 2 presents the numerical parameters that define *ParticleWorks*-supported domains and describes the significance of each numerical argument used in the animation statements.

In addition to describing the source (i.e., initial positions) of new particles, *ParticleWorks* uses domains to describe the ranges for other stochastically selected particle properties. For instance, the initial velocities of particles may also be described with a domain. In this case, the velocity vector is conceived as having its tail at the origin and its tip at a random point inside the specified domain, thus specifying both the magnitude and direction. For instance, consider the indicated initial velocities for the linear source domain in Fig. 6(a). To specify initial velocities of the nature indicated, another linear domain can be used. Fig. 7(a) presents a cross-sectional view of the linear source domain presented in Fig. 6(a). The velocity domain in this case is another straight line on which all the velocity vector tips (arrowheads) lie. Generated particles randomly choose a point in this domain to select their initial velocity (magnitude and direction). This velocity domain could also be specified as a rectangle instead of a line if greater variability in magnitude (but not in direction) were desired. This scenario is presented in Fig. 7(b). If all emitted particles in this example were to have the same velocity and the same direction, then the velocity domain is simply a single point (i.e., a point domain).

In addition to positions and velocities, domains are also useful in stochastically selecting the initial color of emitted particles. *ParticleWorks* is implemented in OpenGL (Woo et al. 1997) wherein the full color space is defined by the zero to one (0.0 to 1.0) range along the red, green, and blue axes. For instance, a color represented as (0,0,0) indicates pure black, (1,1,1) indicates pure white, (1,0,0) is pure red, etc. The result of specifying particle colors as a domain is that each created particle can select a random color that lies in that domain. For instance, a linear domain between the points (1,0,0) and (1,1,0) will choose points on a line between red and yellow, giving each new particle a random color on that line.

This property can be used in interesting ways to depict the realistic appearance of construction materials that cannot be described by a single, solid color. For instance, in Fig. 2, the third

Table 2. Domain Definition Parameters

Domain	Parameters	Significance
Point	x, y, z	Single point
Line	$x1, y1, z1, x2, y2, z2$	End points of a line segment
Triangle	$x1, y1, z1, x2, y2, z2, x3, y3, z3$	Vertices of a triangle
Plane	ox, oy, oz, nx, ny, nz	Point o is on the plane; n is a normal vector to the plane
Rectangle	$ox, oy, oz, ux, uy, uz, vx, vy, vz$	Point o is on the plane; u and v are nonparallel basis vectors in the plane
Box	$x1, y1, z1, x2, y2, z2$	Minima and maxima of an axis-aligned box
Sphere	$ox, oy, oz, radius1, radius2=0.0$	Point o is the center of the sphere; radius1 and radius2 are the outer and inner radii
Cylinder	$x1, y1, z1, x2, y2, z2, radius1, radius2=0$	The two points are the end points of the cylinder's axis; radius1 and radius2 are the outer and inner radii
Cone	$x1, y1, z1, x2, y2, z2, radius1, radius2=0$	The first point is the cone's apex; second is the other end point of the cone's axis; radius1 is the cone's base radius; radius2 is the base radius of another cone to subtract from the first cone to create a conical shell
Disk	$cx, cy, cz, nx, ny, nz, radius1, radius2=0.0$	Point c is the center of the disk in the plane with normal n; radius1 and radius2 are the outer and inner radii
Blob	$x, y, z, stddev$	The point is the center of a normal probability density of standard deviation stddev

statement specifies that the initial color of particles representing the sprayed water be randomly selected from a line that connects the white and light-blue colors in 3D color space. The color of particles representing a stream of mucky water being pumped out of a caisson can similarly be randomly sampled from a domain that contains shades of white, blue, and brown. Such an example is presented later in Fig. 11.

Fluid Object Dynamics

Once the rules for selecting particle attribute values and particle generation are specified, *ParticleWorks'* fuzzy objects begin emitting particles at the indicated rate at each subsequent time instant. The emitted particles originate at the indicated source and select all attributes in accordance with their designated values. Once emitted, *ParticleWorks* treats each particle as a point mass under the influence of prevailing forces. Each existing particle is updated at subsequent time instants according to the physically based analysis procedure discussed in the earlier sections.

In addition, *ParticleWorks* also implements simple collision detection and response of particles in a system. This is necessary to realistically represent many fluid construction materials and processes. For instance, as the animation snapshot in Fig. 5 presents, when concrete is placed into a form, the concrete ''particles'' must be physically constrained by the form panels. Similarly, when gushing water strikes a surface, it is not instantaneously absorbed. Instead, water ''particles'' create a splash whose features depend on physical properties of both water and the surface.

Collision Detection and Response

For each specified impermeable surface (represented by a planar domain), *ParticleWorks* tests whether particles will pass from being outside the illegal domain to being inside it if the next positional update were to be applied at a particular instant. If they would pass through the surface of the domain, they are instead bounced off it. There are thus two parts to this interesting problem: (1) detecting particle-plane collisions, and (2) computing an appropriate response to them. General collision detection in virtual environments is a challenging subject and is a major research theme in computer graphics. However, as the following discussion clarifies, simple particle-plane collisions are relatively easy to detect.

Fig. 8 depicts a dynamic particle moving towards a planar surface. The particle's current position and velocity vectors are given by x and $\vec{v}$, respectively. The point given by vector p lies on the plane. The vector $\vec{n}$ is normal to the plane and points toward the legal side of the surface. Using vector mathematics, a possible particle-plane collision can be detected by computing the vector dot product $(x-p)\cdot\vec{n}$. The value of the dot product helps make the following inferences:

- A value greater than zero indicates that there is no contact and the particle is on the legal side of the barrier,
- A value less than zero indicates that the particle has penetrated the surface and is now on the wrong side of the barrier, and
- An exact zero value indicates that the particle is in perfect contact with the barrier surface.

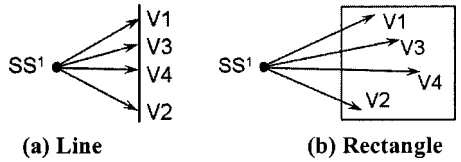

Fig. 7. Velocity domains

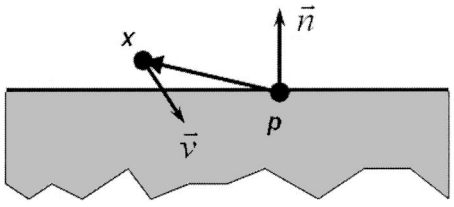

Fig. 8. Detecting particle-plane collision

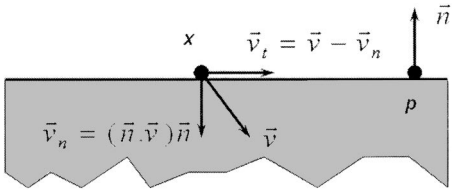

Fig. 9. Incoming velocity vector decomposition

In addition, a particle in perfect contact with the barrier surface may not be colliding with that surface if the particle is moving away from it. The relative velocity of the two bodies must thus be determined by checking the vector dot product $\vec{n}\cdot\vec{v}$. If the value is negative (indicating opposing directions), it can be inferred that the particle is in colliding contact with the surface. When *ParticleWorks* detects a particle-plane penetration or contact, the physical particle response to the collision is computed as follows. The incoming velocity vector of the colliding particle $\vec{v}$ is first decomposed into components normal and tangential to the barrier surface. As Fig. 9 presents, the normal component of vector $\vec{v}$ can be computed as $\vec{v}_n=(\vec{n}\cdot\vec{v})\vec{n}$. The tangential component is then given by $\vec{v}_t=\vec{v}-\vec{v}_n$.

The direction of the normal component is then reversed and multiplied by a coefficient of restitution k_r whose value can be interactively set to simulate collisions of varying elasticity (i.e., to control the particle's bounce). The tangential component is similarly multiplied by $(1-k_f)$, where k_f represents the degree of friction the particle experiences when moving horizontally on the surface. The modified velocity components are then recomposed into a new velocity vector $\vec{v}^1$ heading away from the surface. Fig. 10 presents this procedure graphically.

ParticleWorks provides language statements to set the values of the restitution and friction coefficients. Combinations of elastic, nonelastic, frictionless, and nonfrictionless collisions between particles and rigid surfaces can be described by varying the coefficients of restitution and friction interactively. This can be used to visually simulate interaction between different types of fluid materials and rigid physical constraints. However, since flowing material masses are approximated as a group of individual particles, this method is obviously not accurate physically and is only intended to feign realism in visualizations. For instance, as the animation snapshot of a dewatering process in Fig. 11 indicates, the implementation does not produce particularly accurate visual results for pure liquids that splash rather than bounce.

In addition, *ParticleWorks* does not consider any interparticle collisions and/or interactions between adjoining particles in a particle system. Instead, *ParticleWorks* relies on the stochastic processes that sample the particles' initial properties (i.e., positions and velocities) to accurately describe the overall shape of the intended fluid object volume. For example, in the case of the

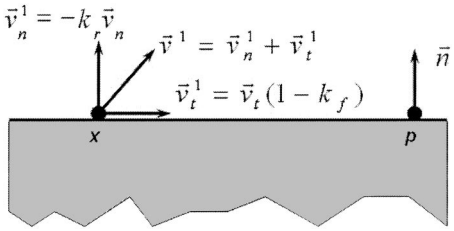

Fig. 10. Outgoing velocity vector decomposition

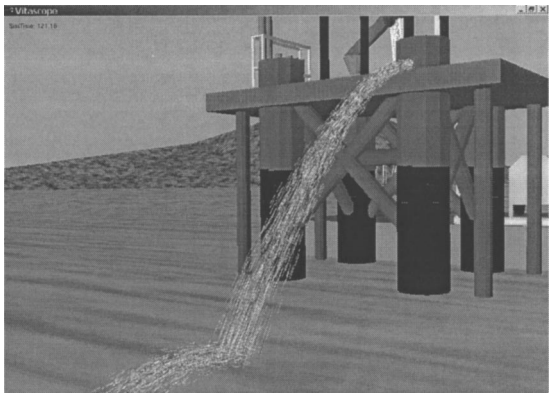

Fig. 11. Animation snapshot of dewatering process

flowing concrete, some pairs of concrete blobs (represented by particles) could potentially penetrate one another or even merge together while accurately maintaining the general shape of the flowing concrete as a whole. Furthermore, a fluid object such as concrete is represented by a particle system only during its motion, after which it is replaced by a traditional, surface-based 3D CAD model. In the current example, concrete is represented as a particle system only during the time when it flows from the bucket and reforms into the shape of the forms on account of the particle-plane interactions. As this reformation takes place, the particles (i.e., concrete blobs) are gradually destroyed (by setting age limits) and the reformed concrete itself is replaced by a solid 3D CAD model having the appearance of concrete and the shape of the forms (e.g., a slab).

In the case of multiple concrete placements at the same location at different time instants (e.g., thick foundation plate), the reformed concrete mass is depicted by a rising, solid, planar surface that ascends by an appropriate amount every time a bucket of concrete is placed. The introduction and manipulation of these solid CAD models to represent reformed fluid objects is achieved separately using core VITASCOPE language statements. The manipulation of the volume of such solid 3D CAD models can be synchronized with the volume of the bucket from which the concrete is being placed and the time required for placement. A solid CAD surface can similarly be used to depict the lowering surface of concrete in a bucket as placement progresses.

Generic Extensible Approach

ParticleWorks' general animation statements allow instantiation and manipulation of generic particle systems. *ParticleWorks* does not provide higher-order statements to directly create specific fluid construction objects such as concrete, water, or slurry. Instead, a generic fluid object is provided, and properties of its constituent particle system are exposed. Keeping *ParticleWorks'* statements general was an important design decision that was made for the significant reason that specificity decreases flexibility and/or user-customizability whereas generality increases it. By keeping *ParticleWorks'* statements general, users have the flexibility to use them in different creative ways. In addition, VITASCOPE (which *ParticleWorks* extends) has a multilayered extensible model. In the first tier, animation statements that extend VITASCOPE may be implemented by concatenating core VITASCOPE language statements. Subsequent tier extensions may use not only VITASCOPE's core statements, but higher-tier

Table 3. Usage of Higher-Order *Particle Works* Statements

Statement	Usage
RAIN [Intensity];	Initialize and place a fuzzy object in the scene setting particle system attributes to resemble rain drops
SNOW [Intensity];	Initialize and place a fuzzy object in the scene setting particle system attributes to resemble snow crystals

extension statements as well. As explained next, this in effect allows any researcher to design and implement higher-level animation statements that describe specific types of fluid objects.

Precipitation

Precipitation such as rain and snow play an important role in construction. The occurrence and intensity of such precipitation may affect outdoor working conditions significantly, most often adversely. Visualizing precipitation can be of significant value in animations of simulated construction processes in that periods of idleness or low productivity resulting from adverse weather conditions can be easily communicated, verified, and validated without any textual or verbal clarification.

The writers concatenated *ParticleWorks'* statements (first-tier VITASCOPE extensions) to design and implement two second-tier extension statements to the VITASCOPE language. These statements, presented in Table 3, allow the use of simple parametric-text commands to describe the type and intensity of existent precipitation on animated construction sites. Each of these statements, when executed, in turn invoke a set of core *ParticleWorks* statements. For instance, the annotated pseudocode that implements the RAIN statement is presented below

```
RAIN [Intensity]
{
    FuzzyObject Cloud [Cover entire jobsite];
    Cloud.SetRate [Intensity];
    Cloud.SetColor [Blue];
    Cloud.SetSink [Ground surface];
    Place Cloud [High above the jobsite];
}
```

A rain cloud is instantiated as a fluid object and placed at a high altitude on the virtual jobsite. The attributes of the cloud's particle system are then set such that the emitted particles resemble rain drops, and their density resembles its intensity. Precipitation of snow is implemented in exactly the same manner, except for the appearance specified to its particles.

Future Work

This research can be extended in several ways. The concept of particle systems can be further explored to describe several other processes common to construction. For instance, blasting operations are common in tunneling and quarrying. Visualizing explosions and the resulting debris caused by such operations is an interesting challenge that could be addressed using adaptations of the particle system concept. In particular, exploring methods to describe such processes in simple, parametric-text statements calls for some very interesting work.

Particle systems can also be adapted to describe bounded, dynamic formations of generic simulation entities. Examples of such entities include vehicles traveling through congested construction work zones and people walking in crowded places. Particles may be represented in virtual worlds as any CAD model (e.g., car, truck, person, etc.). Particle systems may thus present a unique technique for visually describing such bounded formations while simultaneously reducing the number of instructions necessary for their description. Engineers are typically interested in the overall effect of congestion rather than any particular entity (e.g., vehicle or person). Techniques of describing random bounded group formations as particles can thus be of significant help in reducing the complexity and size of discrete-event simulation models that author such visualizations. Literature provides several examples of particle systems adapted to visualize such diverse things as flocking birds, herding sheep, and schooling fish (Reynolds 1987), in addition to pedestrian and road traffic (Schaefer et al. 1998). Designing methods to describe interparticle avoidance and motion resulting from interaction with other particles are interesting research issues to explore in such adaptations.

Future research can also focus on improving the physics behind particle system dynamics. This includes considering additional physical world forces (e.g., adhesion, attraction, repulsion) during analysis and experimenting with improved, more accurate numerical integrations methods. In addition, improved methods of dynamic fluid analysis may be adopted in the physical analysis computations. Methods such as those based on the Navier-Stokes equations perform accurate analysis of motions of viscous liquids in three dimensions (Stam 1999). The perpetual advances in computing power promise to make this work feasible in the near future.

Summary and Conclusions

The presented research extends the state-of-the-art of 3D visualization of simulated construction processes. The work puts in place the technology that DES users in construction can exploit to automatically create dynamic, convincing visualizations of construction processes that involve unstructured, fluid materials. The design of techniques to accurately animate the shape, form, motion, and dynamic behavior of unstructured, fluid construction materials in simulation-driven animated 3D virtual construction worlds requires three key technologies: (1) computationally feasible physical analysis to accurately describe the dynamics of fluid material volumes; (2) simple, parametric animation methods (i.e., statements) with semantics rich enough to convert terse and discretely recorded animation descriptions into smooth motion; and (3) efficient rendering techniques to represent the descriptions in 3D virtual worlds such that the results look visually convincing, move realistically, and can be animated in real time.

Classical techniques of modeling fluid behavior are highly accurate but require very intensive computation. To achieve a compromise between visual realism, physical accuracy, and reasonable computational times, the presented work treats fluid volumes as clouds of individual point masses whose dynamics are described using an elementary subset of Newtonian mechanics.

The approach has several advantages. First, it allows the determination of the finite subset of discrete static and dynamic components that contributes to the overall appearance and motion of arbitrary fluid material volumes. This guides the design of animation methods (i.e., statements) that achieve smooth, continuous animation based on discrete information recorded at arbitrary, nonfixed time steps. This is critical since external animation authoring processes such as DES models can communicate with other processes only at discrete, but possibly random, sets of

simulated time points. In addition, the adopted approach allows the representation of volumes of dynamic, fluid construction materials in 3D virtual worlds using classical computer graphics techniques based on the particle system concept. The visually appealing results that are obtained demonstrate that the approach is not only possible, but also quite effective.

Acknowledgments

The writers gratefully acknowledge the support of the National Science Foundation (CAREER and ITR programs) to the presented work. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the writers and do not necessarily reflect the views of the National Science Foundation.

References

Burden, R. L., and Faires, J. D. (1989). *Numerical analysis*, 4th Ed., PWS-Kent Publishing Company, Boston.

Carlson, M., Mucha, P. J., Van Horn, R. B., and Turk, G. (2002). ''Melting and flowing.'' *Proc., ACM SIGGRAPH Symposium on Computer Animation*, Association for Computing Machinery, New York, 167–174.

Dorsey, J., Pedersen, H. K., and Hanrahan, P. (1996). ''Flow and changes in appearance.'' *Proc., 23rd Annual Conf. on Computer Graphics and Interactive Techniques*, Association for Computing Machinery, New York, 411–420.

Farr, S. D., and Sisti, A. F. (1994). ''Visualization of general-purpose simulation results.'' *Proc., 4th IEEE Dual Use Technologies and Applications Conference*, IEEE, Piscataway, N.J. Available at ⟨http://www.rl.af.mil/tech/papers/ModSim/DU94.html⟩ (Oct. 10, 2002).

Foster, N., and Fedkiv, R. (2001). ''Practical animation of liquids.'' *Proc., 28th Annual Conf. on Computer Graphics and Interactive Techniques*, Association for Computing Machinery, New York, 23–30.

Foster, N., and Metaxas, D. (2000). ''Modeling water for computer animation.'' *Communications of the ACM*, Vol. 43, No. 7, Association for Computing Machinery, New York, 60–67.

Kamat, V. R., and Martinez, J. C. (2001). ''Visualizing simulated construction operations in 3D.'' *J. Comput. Civ. Eng.*, 15(4), 329–337.

Kamat, V. R., and Martinez, J. C. (2002). ''Scene graph and frame update algorithms for smooth and scalable 3D visualization of simulated construction operations.'' *J. Comput.-Aided Civ. Infrastruct. Eng.*, 17(4), 228–245.

Kamat, V. R., and Martinez, J. C. (2003a). ''Validating complex construction simulation models using 3D visualization.'' *Systems analysis modelling simulation*, Vol. 43, No. 4, Taylor and Francis Group, London, 455–467.

Kamat, V. R., and Martinez, J. C. (2003b). ''Automated generation of dynamic, operations level virtual construction scenarios.'' *Electronic journal of information technology in construction*, (ITcon), Vol. 8, special issue on virtual reality technology in architecture and construction, Royal Institute of Technology, Stockholm, Sweden, 65–84.

Kamat, V. R., and Martinez, J. C. (2003c). ''*ParticleWorks* Add-On for VITASCOPE Reference.'' Dept. of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Va. Available at ⟨http://strobos.cee.vt.edu/vitascope/Dissertation/ParticleWorksReference.pdf⟩ (Apr. 20, 2003).

Kamat, V. R., and Martinez, J. C. (2003d). ''VITASCOPE Language Reference.'' Dept. of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Va. Available at ⟨http://strobos.cee.vt.edu/vitascope/Dissertation/VitascopeReference.pdf⟩ (Apr. 20, 2003).

Koo, B., and Fischer, M. (2000). ''Feasibility study of 4D CAD in commercial construction.'' *J. Constr. Eng. Manage.*, 126(4), 251–260.

Law, A. M., and Kelton, W. D. (2000). *Simulation modeling and analysis*, 3rd Ed., McGraw-Hill, New York.

Lipman, R., and Reed, K. (2000). ''Using VRML in construction industry applications.'' *Proc., 5th Symposium on Virtual Reality Modeling Language (Web3D-VRML)*, Association for Computing Machinery, New York, 119–124.

McAllister, D. K. (2001). *The design of an API for particle systems*, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, Chapel Hill, N.C. Available at ⟨ftp://ftp.cs.unc.edu/pub/publications/techreports/00-007.pdf⟩ (Oct. 10, 2002).

Op den Bosch, A. (1994). ''Design/construction processes simulation in real-time object-oriented environments.'' PhD dissertation, Georgia Institute of Technology, Atlanta.

Paramount (1982). *Star Trek II: The Wrath of Khan* (film), Hollywood, Calif. Available at ⟨http://www.siggraph.org/education/materials/HyperGraph/animation/movies/genesisp.mpg⟩ (Oct. 10, 2002).

Park, B. (2002). ''Development of a virtual reality excavator simulator: A mathematical model of excavator digging and a calculation methodology.'' PhD dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Va.

Peurifoy, R. L., and Schexnayder, C. J. (2002). *Construction planning equipment and methods*, 6th Ed., McGraw-Hill, New York.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*, 2nd Ed., Cambridge University Press, New York.

Reeves, W. T. (1983). ''Particle systems—A technique for modeling a class of fuzzy objects.'' *Comput. Graph.*, 17(3), 359–376.

Reynolds, C. W. (1987). ''Flocks, herds, and schools: A distributed behavioral model.'' *ACM SIGGRAPH Comput. Graph.*, 21(4), 25–34.

Rohrer, M. W. (2000). ''Seeing is believing: The importance of visualization in manufacturing simulation.'' *Proc., 2000 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, 1211–1216.

Schaefer, L. A., Mackulak, G. T., Cochran, J. K., and Cherilla, J. L. (1998). ''Application of a general particle system model to movement of pedestrians and vehicles simulation.'' *Proc., 1998 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, 1155–1160.

Stam, J. (1999). ''Stable fluids.'' *Proc., 26th Annual Conf. on Computer Graphics and Interactive Techniques*, Association for Computing Machinery, New York, 121–128.

Tsay, T. C., Hadipriono, F. C., and Larew, R. E. (1996). ''Virtual reality modeling for bridge construction.'' *Proc., 3rd Congress on Computing in Civil Engineering*, ASCE, Reston, Va., 63–69.

Witkin, A., and Baraff, D. (2001). ''Physically based modeling.'' SIGGRAPH 2001 Course Notes. Available at ⟨http://www.pixar.com/companyinfo/research/pbm2001/index.html⟩ (Oct. 10, 2002).

Woo, M., Neider, J., and Davis, T. (1997). *OpenGL programming guide*, 2nd Ed., Addison-Wesley, Reading, Mass.