Engineering with Computers (2007) 23:79–91
DOI 10.1007/s00366-006-0046-6

ORIGINAL ARTICLE

Interactive collision detection in three-dimensional visualizations of simulated construction operations

Vineet R. Kamat · Julio C. Martinez

Received: 6 April 2006 / Accepted: 24 August 2006 / Published online: 11 October 2006

Abstract This paper presents research that led to the design and implementation of fast and interactive collision detection methods that can be used to identify and report undesirable conflicts that occur among static (e.g., structure in-place, idle equipment), dynamic (e.g., active machines and workers), and abstract (e.g., hazard spaces) construction resources in 3D animations of construction operations modeled using discrete-event simulation. Computational efficiency and interactive speed in the designed interference detection methods were the primary challenges that the research addressed. In addition, the efficiency and speed were achieved with minimal trade-off against accuracy. In order to achieve this, the authors capitalized on advanced documented algorithms for proximity queries between arbitrarily moving 3D geometric objects to design mechanisms for interference detection, control, and response in construction process visualizations. The designed methods are implemented in a software tool called C-Collide that integrates as an add-on with the VITASCOPE visualization system.

Keywords Animation · Discrete-event simulation · Interference detection · Simulation · Validation

V. R. Kamat (✉)
Department of Civil and Environmental Engineering,
University of Michigan, 2340 GG Brown,
2350 Hayward Street, Ann Arbor, MI 48109-2125, USA
e-mail: vkamat@umich.edu

J. C. Martinez
Department of Civil and Environmental Engineering,
Virginia Polytechnic Institute and State University,
200 Patton Hall, Blacksburg, VA 24061-0105, USA
e-mail: julio@vt.edu

1 Introduction

Visualization of simulated construction operations can be of significant help in verifying and validating discrete-event construction process models. In addition, visualization can provide valuable insights into subtleties of planned construction operations that are otherwise non-quantifiable and non-presentable. The necessity to effectively communicate modeled construction processes and the resultant evolving constructed facilities is the motivation behind the authors' recent visualization research efforts [9–12]. These efforts focused on designing automatic, process simulation-driven methods to visualize construction processes and the resulting products in dynamic 3D worlds.

The tangible outcome of this research is the VITASCOPE visualization system. VITASCOPE is an acronym for VIsualizaTion of Simulated Construction OPErations. VITASCOPE is a user-extensible 3D animation language designed specifically for visualizing modeled construction operations in smooth, continuous, dynamic 3D virtual worlds [8]. A limited subset of the VITASCOPE language and the corresponding prototype were referred to as the Dynamic Construction Visualizer (DCV) in prior publications [9, 10]. VITASCOPE and its add-ons, documentation, and solved examples are available for download from http://www.pathfinder.engin.umich.edu.

1.1 Main contribution

In this paper, an extension to the VITASCOPE visualization system is described. The paper presents a tool, C-COLLIDE that can identify and report any and all undesirable conflicts that can occur among static

(e.g., structure in-place, idle equipment), dynamic (e.g., active machines and workers), and abstract (e.g., hazard or protected spaces) construction resources in dynamic 3D construction process visualizations.

Common types of clashes that can occur on real construction sites and that C-COLLIDE can identify beforehand in process visualizations include (1) intersection among physical in-place components (i.e., design interferences), (2) intersection among in-place components, components in transit, and/or pieces of moving equipment during construction (i.e., constructability interferences), (3) craft interferences and accidents e.g., collision between two pieces of equipment operating in the same area, and (4) space intrusions e.g., any resource (worker or equipment) encroaching arbitrarily shaped hazard or protected areas of the jobsite.

The research capitalized on advanced documented algorithms for efficient proximity queries between arbitrarily moving 3D geometric objects to design mechanisms for interference detection, control, and response in 3D construction process visualizations. C-COLLIDE's interference detection capabilities dynamically check each motion of VITASCOPE scene objects to determine if any pairs of scene objects interfere undesirably. This provides users with a clear understanding of all object motions and potential interferences in any area of activity on a simulated construction job site.

Previous studies in construction have applied collision detection algorithms in various static and dynamic graphical visualization contexts. Applications such as Interference Manager [3] and the Interference Detection Language (IDL) [5] use collision detection algorithms to identify geometric clashes between permanent facility components and automatically verify the design of a proposed facility. Akinci et al. [1] have used interference detection algorithms to check for intersections among rectangular prisms (representing spaces required by construction activities) to perform time-space conflict analyses in 4D CAD visualizations. Finally, tools such as Dynamic Animator (Bentley 2] utilize collision detection algorithms to check for geometric interferences between objects during playback of manually created motion sequences.

Neither of the above applications, however, is time sensitive. In other words, in all the abovementioned applications, there is no relation between elapsed simulation time and elapsed wall clock time. Design verification operates on static CAD data and obviously has no notion of time associated with the computation. Although time–space conflict analysis identifies spatio-temporal clashes among construction spaces, there is no relation between simulation time (i.e., activity durations) and elapsed wall clock time i.e., the computation is performed in batch processing mode. In Dynamic Animator, the elapsed visualization time is related to the number of screen updates rather than wall clock time. Thus, in all these applications, efficiency in interference detection computations, although desirable, is not of the essence.

VITASCOPE animations are time sensitive because elapsed visualization time is always related to the elapsed wall clock time by a distinct viewing ratio [10]. This viewing ratio is strictly maintained regardless of the number of screen updates between subsequent simulation time instants. Thus, to depict smoothly moving scene objects and to avoid jerkiness in the presented graphics, VITASCOPE must always update the screen at an acceptable refresh rate (ideally > 24 frames/s). The impact of any collision detection computations on this frame rate must be minimal.

Efficiency in the adopted proximity query algorithms is therefore of paramount importance in C-COLLIDE. In addition, this efficiency cannot be traded off against accuracy. C-COLLIDE's main contributions are as follows:

- Highly efficient, interactive, and accurate methods for automated construction process level interference detection and conflict analyses.
- Integrated framework for performing combined design level, activity level, and process level spatio-temporal interference analyses.

2 Importance of the research

Construction process visualization involves the faithful representation of real-world construction operations and the resulting evolving products in smooth, dynamic, continuous 3D virtual worlds. The types of undesired collisions and interferences that can occur in such visualizations encompass the entire range of undesired interferences that can occur on real construction sites. In 3D visualizations that describe process-level construction details, the goal of collision detection and interference analysis is thus, in general, the identification and reporting of any and all undesirable conflicts that might occur among static, dynamic, and abstract construction resources on any part of the virtual jobsite.

Collision detection and interference analysis in virtual environments by definition requires intensive computation. It is traditionally considered a bottleneck

in smooth, continuous, animated 3D virtual environments [14]. The complexity of the required computation is directly proportional to the complexity of the 3D graphical scene database (i.e.. number and types of objects contained). Virtual worlds describing dynamic construction processes epitomize the description of a complex 3D scene. Construction is performed in an arbitrarily complex dynamic environment where equipment and materials operate together, are attached to one another or taken apart, and the landscape itself changes shape. These actions in turn cause a construction product facility (e.g., building, bridge, tunnel, etc.) to steadily evolve with the passage of time. In such circumstances, the dynamic 3D graphical database that describes construction processes and the resulting products in a virtual world can become extremely large and complex.

A virtual world representing a physical world environment such as a construction site generally consists of N moving objects and M stationary objects, where both N and M can be arbitrarily large. Each object can have an arbitrary shape that is often dynamic and cannot be predetermined. The detection of object to object collisions in such environments is a computationally intensive task because each of the N moving objects can theoretically collide with the other moving objects, as well as the stationary ones. As such, the collision detection test has to be performed at each time step (i.e.. frame) in the visualization—an operation whose computation cost increases sharply with scene complexity. Efficiency in contact determination algorithms is thus paramount in such environments.

This requirement for extreme efficiency is distinct from requirements of many other applications of collision detection algorithms (e.g., design verification from static CAD data) where the computation time is not at as much premium as is in dynamic animation. In dynamic animation, collision tests must be performed on an arbitrary number of objects at each frame in a fraction of time such that the involved computation has minimal adverse influence on the animation's frame rate. In addition and notwithstanding the speed requirement, performed collision tests must be spatially accurate and must often determine the exact point(s) of contact between interfering pairs of scene objects. This can be helpful in contemplating remedial actions to avoid the occurring interferences.

In virtual environments representing physical world processes, interference detection computation can almost always be optimized using contextual rules and assumptions [14]. For example, the bucket, stick, boom, cabin, and the tracks of a single excavator are represented as a hierarchy of distinct 3D objects that are each capable of being manipulated separately inside the virtual world. However, it would be a waste of computational effort if collisions between pairs of all components on the same excavator (e.g., boom and stick, cabin and tracks, etc.) are computed at each frame. Computation effort would similarly be wasted if CAD objects representing resources guaranteed to be non-interacting are repeatedly tested for intersections. Examples of such resources include excavators operating at different distant loading areas and trades operating on different levels of a building that are a few floors apart. Users animating construction processes must thus be able to precisely specify and control the pairs of scene objects that must be tested for possible interferences at each frame. This is essential if the requirement of extreme efficiency in collision detection computations is to be achieved.

2.1 Technical requirements

In order to satisfy the required criteria (functionality, speed, accuracy, and flexibility) imposed by dynamic 3D virtual construction worlds, methods designed for collision detection and interference analysis must be:

- *General* : A dynamic virtual construction site can consist of several objects of arbitrary shapes, forms, and sizes (from the collision detection algorithm's viewpoint). The algorithms adopted to perform interference analysis in dynamic construction process visualizations must therefore operate on 3D CAD models of arbitrary shapes, sizes, and forms. In addition, the algorithms must not make any assumptions about object motions, velocities, and accelerations.
- *Efficient*: On a virtual construction site with N mobile and M stationary entities, monitoring $N + M$ objects and checking $\binom{N}{2} + NM$ pairs of objects for intersection at every time step can become time consuming and inefficient as N and M get large (typical of virtual construction sites). Visualizations describing modeled operations must be smooth and continuous and not jerky [13]. To achieve interactive rates of interference analysis, then, the total number of pair-wise intersection tests must be reduced before performing exact collision tests on the object pairs that are in the close vicinity of each other. Contextual rules and assumptions are one possible technique. In addition, literature describes several algorithmic optimizations to deal with large datasets of possibly colliding objects [4].

- *Accurate*: When an undesirable collision or interference occurs between two virtual entities involved in construction, users are typically interested in the exact point(s) and/or location(s) of contact. This is especially true if the involved objects are occluded by other objects or if the interference is subtle. Feedback on the exact details of the occurring interference can further be of help to users in devising strategies to avoid interference during actual construction. The adopted algorithms must thus report accurate details of any detected collisions and interferences without overloading the computation pipeline.
- *Dynamic*: A 3D virtual construction site is represented as a scene graph inside the computer [10]. Since construction operations are performed in an inherently dynamic environment, scene objects (i.e., resources) move, and may appear or disappear at any time during visualization (when they travel to and out of users' visual area of interest). The data structures for collision detection and interference analysis must be able to handle this dynamic behavior. It must be possible to easily insert and remove scene objects from the collision detection engine's list of monitored scene objects.
- *Interactive*: Users must be able to selectively specify the scene objects that the collision detection engine must monitor and the nature of the feedback that must be generated when interferences are detected.

3 Collision detection algorithms

Collision detection between 3D geometric models (also known as contact determination or interference detection) has been extensively studied in fields such as geometric modeling, computer graphics, robotics, and computational geometry. Since collision detection is employed for a wide variety of applications, different domain specialized algorithms and methods have been developed for the purpose [14].

3.1 Problem domain

Determining interferences among multiple complex moving objects in 3D virtual environments has become a very popular research topic in computer graphics [15]. The primary reason for this research emphasis is the need for extreme efficiency in interference detection in dynamic complex virtual environments. The presented discussion of collision detection algorithms is only limited to this latter context where much of the research addresses the familiar *N*-body problem that signifies the worst case scenario of *N* unconstrained 3D objects moving arbitrarily in a scene containing possibly M other stationary objects. The non-emergence of a universally popular efficient algorithm is the primary motivator for continued ongoing research in detecting collisions between objects in such dynamic virtual environments.

The collision detection problem in a dynamic 3D virtual world consisting of many moving and stationary objects can be generally separated into three distinct areas:

– Detecting the occurrence of a collision between a pair of scene objects
– Determining the point of contact to some degree of accuracy
– Deciding the nature of the response to the detected collision

The first two steps involve geometric computation and are generally the focus of core research in computer graphics. The last step, i.e., collision response is usually application dependent and requires a dynamic model.

3.2 Relevant taxonomy

The algorithms to accomplish the first two steps of detecting collisions between 3D geometric objects can be broadly classified into the following two categories based upon their computation scheme:

– *Two-phase methods*: These algorithms divide the contact determination stage into two distinct phases each time a check is desired. The first broad phase quickly culls or eliminates from further computation pairs of scene objects that cannot possibly collide. Approximate geometric techniques, specific rules and context of the pertinent visualization, and/or algorithmic optimization techniques are used for this purpose. Pairs of scene objects that pass the first broad phase are deemed to be possibly colliding. These object pairs are then subjected to accurate detailed collision detection analysis using suitable low-level geometric techniques. This second narrow phase generally requires much more geometric computation.
– *Single-phase methods*: Algorithms in this class do not utilize the first broad object elimination phase. Instead, they directly subject each existing scene object pair to detailed, narrow-phase geometric tests to determine any existing contact. By eliminating the broad phase altogether, such algorithms often prove efficient in cases where most scene objects are

generally deemed to be possibly colliding and must be checked in detail anyways. In such cases, the broad phase would add unnecessary computation overhead.

In addition to computation schemes, collision detection algorithms may also be classified based upon several different criteria such as the input geometry type (e.g., polygonal, non-polygonal, structured, unstructured, etc.), types of queries supported (e.g., contact determination, exact separation distance, depth of penetration, etc.), and types of consumer visualization environments (e.g., scientific visualizations, games, interactive simulators, etc). We refer the reader to Lin and Gottschalk [14] for a comprehensive and informative survey on these taxonomies and references to several works on core collision detection research.

3.3 Algorithmic optimizations

All algorithms designed for real-time contact determination among several moving objects in virtual environments generally exploit spatial and temporal coherence among scene objects in their computation. Moving objects in 3D worlds generally tend to occupy the same region of space during subsequent time instants. In other words, the amount by which most objects move each frame is generally less compared to the dimensions of the objects themselves.

Several algorithms exploit this characteristic that is common of virtual environments representing physical world processes. Some algorithms require bounds on the motion of the objects (e.g., object velocities or accelerations). Other algorithms such as the ones based on interval arithmetic need a closed form expression of the motion as a function of time. Some algorithms demand no information on the motion but need only the placements of the objects at successive time steps.

4 Technical approach

The authors have adopted a two-phase, hierarchical multi-level approach presented in Hudson et al. [7] to address the problem of collision detection and interference analysis in dynamic 3D construction process visualizations. From a collision detection algorithm's viewpoint, a typical virtual construction site is composed of N arbitrarily moving and M stationary objects. Each of the N objects can possibly collide with the other N-1 objects as well as with the M stationary objects. However, since activities on a construction site are typically spread both laterally and vertically, not all object pairs can be deemed to be possibly interacting at any time instance. The two-phase approach described in the previous section is thus more suitable and efficient for this research because during each intersection test (one per frame), pairs of scene objects that are not within interacting distance of each other can be quickly eliminated. This reduces the computation load manifold by limiting detailed pair-wise intersection tests to only those object pairs that are very near each other.

Figure 1 presents the architecture of the adopted interference detection algorithm. After each dynamic scene update, a quick conservative approximate test based on loosely fitting, imaginary bounding volumes known as axis-aligned bounding boxes (AABBs) first finds potentially colliding pairs of objects among the set of existent scene entities. The algorithm adopts a procedure known as an N-body sweep and prune in computer graphics literature [4]. After marking pairs of potentially colliding objects, an exact two-level test computes whether the two objects in each marked pair actually collide.

In this stage, the algorithm constructs another kind of imaginary bounding volumes called oriented bounding boxes (OBBs) around each scene object and its geometric primitives. These bounding volumes are then organized into a hierarchical structure called OBBTrees [6]. Overlap tests on the constructed OBBTrees followed by exact geometric primitive intersection tests are then performed to confirm collisions and to determine the exact point(s) of contact. The following subsections clarify all of the above terminology and describe each of the geometric computation phases of the algorithm. Discussion of the analysis and response to detected interferences is deferred to the following section.

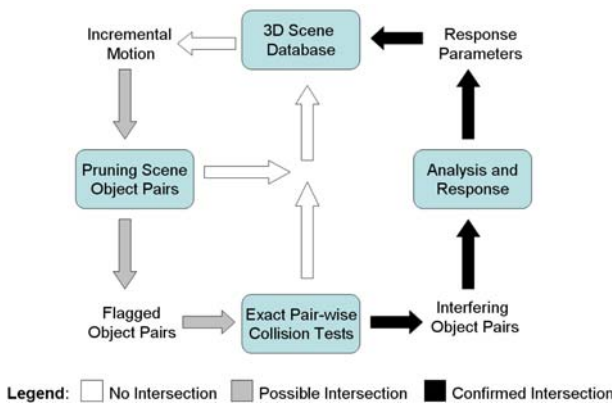

Fig. 1 Interference detection algorithm architecture

4.1 Pruning scene object pairs

The sweep and prune part of the algorithm reduces the number of computationally intensive exact pair-wise collision tests by eliminating from the computation pipeline pairs of scene objects that are far apart. This is accomplished by constructing imaginary, loosely fitting 3D bounding volumes to surround (i.e., enclose) each scene object and then determining which pairs of those volumes overlap. Only pairs of objects whose loose-fitting bounding volumes intersect (indicating that the objects are in close vicinity and possibly colliding) are passed on to the next stage of exact contact determination computation.

In the sweep and prune phase, the adopted algorithm calculates easy-to-compute rectangular bounding volumes called axis-aligned bounding boxes (i.e., AABBs) to enclose each scene object. AABBs are imaginary rectangular prisms whose sides are aligned with the three major coordinate axes irrespective of the enclosed object's orientation in global space. The orientation of the enclosed object affects the size and the amount of free space inside the enclosing bounding box. As a result, AABBs describe loosely fitting bounding volumes for all but the most ideal enclosed object orientation. Figure 2 illustrates this by projecting in two dimensions.

The constructed bounding boxes are analyzed in 3D space to determine overlapping pairs. A dimension reduction approach is used for the purpose [4]. The sweep and prune algorithm begins by projecting each three-dimensional bounding box onto the three coordinate axes (X, Y, and Z). Since the bounding boxes are axis aligned regardless of the enclosed object orientation, their projection onto the coordinate axes describes finite linear intervals. Figure 3 presents this graphically for the *X*-axis. By determining overlaps among the projected intervals of bounding box pairs, it is possible to deduce whether or not the boxes overlap in 3D space. As is apparent from Fig. 3, this follows from the simple intuitive result that a pair of AABBs can intersect in 3D if and only if their projected intervals overlap in all three dimensions.

At each time instant (i.e., frame) during visualization, the sweep and prune algorithm reconstructs the AABBs for each scene object and flags pairs of scene objects whose AABBs intersect. The remaining object pairs are deemed as not colliding and quickly eliminated from the computation pipeline. Only pairs of scene objects that pass the sweep and prune test are passed on to the next detailed collision detection phase that is now described in the following subsection.

4.2 Exact pair-wise collision tests

The next phase of the algorithm detects exact pair-wise collisions among pairs of scene objects that are flagged as potentially colliding by the preceding approximate test. This is accomplished by building hierarchical representations of another type of imaginary 3D bounding volumes called oriented bounding boxes (i.e., OBBs) to enclose objects. OBBs are imaginary rectangular prisms aligned at an arbitrary orientation in 3D space. Unlike AABBs that are always aligned with the co-ordinate axes, OBBs can be oriented in any direction such that the resulting volume encloses the bounded object as tightly as possible. Since OBBs are constructed taking an object's orientation into account, the size and the amount of free space (i.e., the tightness) inside the enclosing bounding box always remain constant. As a result, OBBs describe tight fitting bounding volumes for all enclosed object orientations. Figure 4 illustrates this graphically by projecting a set of OBBs in two dimensions.

4.3 Construction of bounding volume hierarchy

The exact collision test algorithm computes a tree of OBBs (called OBBTree) for every object, with a box containing the entire object as the root and boxes containing only one or a very few primitives of the object as the leaves. The tree construction process, presented graphically in Fig. 5, has two components. First is the placement of a tight fitting OBB around each (sub)object (represented as a collection of

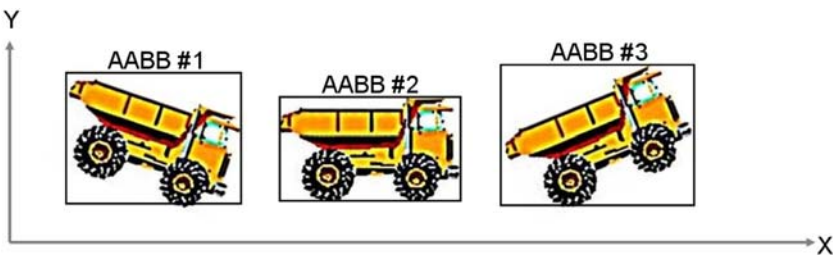

Fig. 2 Axis aligned bounding boxes

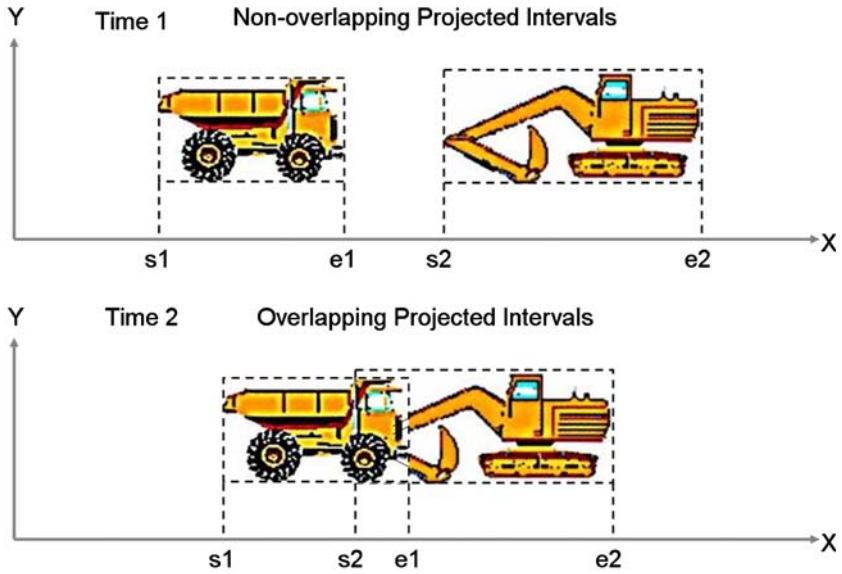

Fig. 3 Bounding box projections

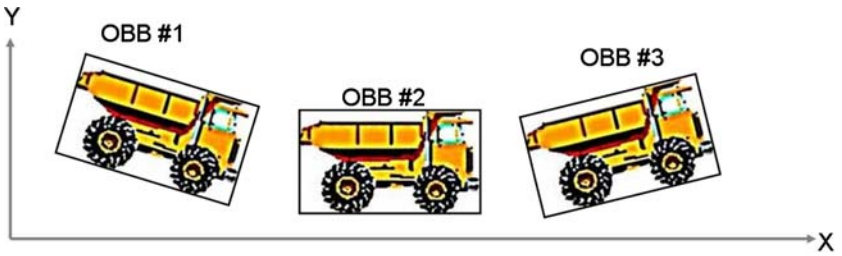

Fig. 4 Oriented bounding boxes

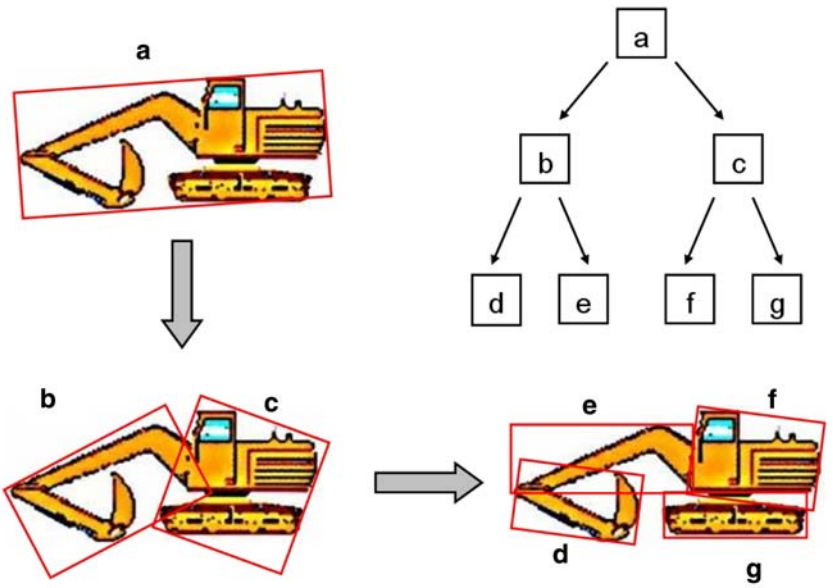

Fig. 5 Building the OBBTree

polygons), and second is the grouping of nested OBBs into a tree hierarchy.

The algorithm approximates the collection of polygons in a (sub)object with an OBB of similar dimensions and orientation. The OBB computation procedure makes use of first- and second-order statistics summarizing the vertex coordinates. They are the mean and the covariance matrix, respectively. The eigenvectors of a symmetric matrix, such as the covariance matrix, are mutually orthogonal. After

normalizing them, they are used as a basis. The procedure finds the extremal vertices along each axis of this basis, and sizes the bounding box, oriented with the basis vectors, to enclose those extremal vertices. Since two of the three eigenvectors of the covariance matrix are the axes of maximum and of minimum variance, they tend to align the box with the geometry of a tube or a flat surface patch. The exact formulae and construction details of this procedure are described in detail in Gottschalk et al. [6].

To construct the OBBTree for a scene object, the algorithm adopts a top-down recursive approach that partitions the primitives in each box into two sub-boxes, based on the location of their centers. The procedure begins with the group of all polygons (i.e., the OBB of the entire object), and recursively subdivides that OBB until all leaf nodes are indivisible. The subdivision rule that is adopted in this procedure splits the longest axis of a box with a plane orthogonal to one of its axes, partitioning the polygons according to which side of the plane their center point lies on. The subdivision coordinate along that axis is chosen to be that of the mean point of the vertices. If the longest axis cannot be subdivided, the second longest axis is chosen. Otherwise, the shortest axis is used. The process continues until the group of polygons cannot be further partitioned along any axis by this criterion (i.e., the group is considered indivisible).

4.4 Hierarchical exact overlap test

To check for collision between a pair of objects, the algorithm descends their OBB hierarchies to find any leaf boxes which overlap, and then performs exact intersection tests between the triangles in the overlapping leaves. The root of an OBBTree encloses the entire object itself and its leaves contain one or more primitive constituents. The hierarchy thus defines an intra-object spatial partitioning. Successive levels of OBBs in two OBBTrees must only be tested for intersection if their respective parent OBBs intersects. Figure 6 presents the pseudo code for traversing the constructed bounding volume hierarchies to detect collisions.

The algorithm confirms whether two scene objects are disjoint or not by projecting their OBBs onto some axis (not necessarily a coordinate axis) in space. Akin to the projection of AABBs onto the coordinate axes, the axial projection of each box on an arbitrary axis forms an interval on that axis. If the intervals do not overlap, then the axis is called a ''separating axis'' for those boxes. The existence and identification of a separating axis confirms that the tested boxes are

```
Beginning at the root nodes of two given trees
1.  Check for intersection between two parent nodes
2.  If there is no intersection between two parents
3.        Then stop and report "no collision"
4.  Else check all children of one node against all
    children of the other node
5.  If there is intersection between any children
6.        Then If at leaf nodes
7.             Then report "possible collision"
8.        Else go to Step 4
9.  Else stop and report "no collision"
```

Fig. 6 Traversing bounding volume hierarchies to detect collisions

disjoint. If the intervals do overlap however, then the boxes may or may not be disjoint and further testing is required. Figure 7 graphically presents these scenarios.

The adopted algorithm makes use of the separating axis theorem presented in Gottschalk et al. [6] to check for overlaps between OBBTrees of scene objects. According to the theorem, two convex polytopes in 3D (OBBs in this case) are disjoint if and only if there exists a separating axis orthogonal to a face of either polytope or orthogonal to an edge from each polytope. Each box has three unique face orientations, and three unique edge directions. This leads to 15 potential separating axes to test (three faces from one box, three faces from the other box, and nine pair-wise combinations of edges). If the polytopes are disjoint, then a separating axis exists, and one of the 15 axes mentioned above will be a separating axis. If the boxes overlap, then clearly no separating axis exists. So, testing the 15 given axes is a sufficient test for determining overlap status of two OBBs.

If pairs of leaf OBBs are found to be intersecting during the OBBTree traversal, the algorithm performs exact geometric intersection tests between the primitive triangles in the overlapping leaf OBBs. This final computationally demanding test confirms beyond doubt whether the two scene objects actually intersect and if affirmative, also determines the exact point(s) of contact. This last test is performed only if pairs of scene objects pass the earlier tests that are performed in an order of increasing accuracy (and computation cost). By adopting this hierarchical, multi-level procedure, the algorithm is thus able to provide exact collision detection capabilities while maximizing computational efficiency.

The next section describes the authors' implementation of the adopted algorithms. The implementation is a software tool that provides users with comprehensive, accurate, and interactive interference detection capabilities in dynamic 3D construction process visualizations.

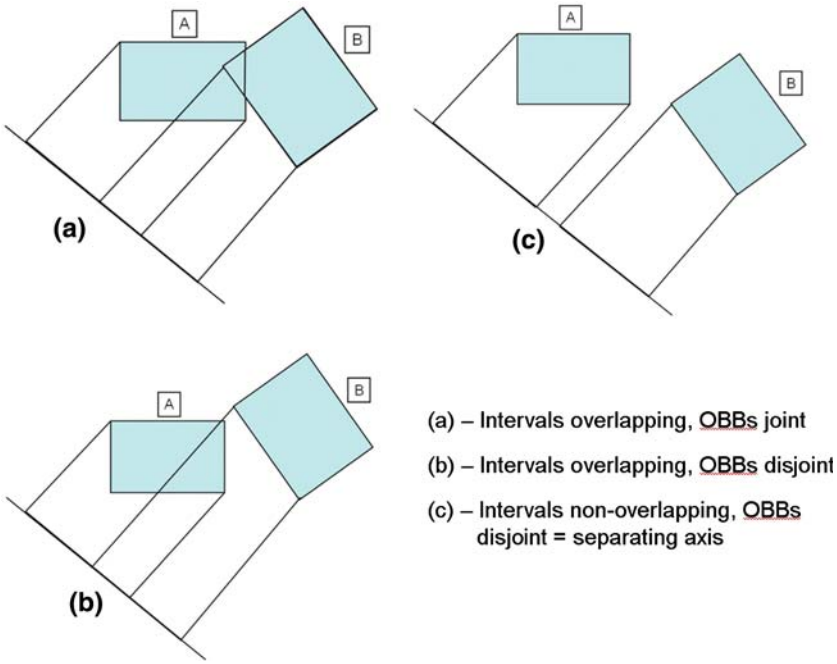

Fig. 7 Searching for a separating axis

5 C-COLLIDE

C-COLLIDE is a software tool that provides users with comprehensive and accurate feedback on any and all undesired interferences that occur among static (e.g., structure in-place, idle equipment), dynamic (e.g., active machines and workers), and abstract (e.g., hazard or protected areas) construction resources in dynamic 3D construction process visualizations. C-COLLIDE is implemented as an extension (add-on) to the VITASCOPE visualization system.

The C-COLLIDE add-on partly redefines as well as extends the VITASCOPE animation language. The add-on redefines several core VITASCOPE animation language statements. The redefined statements instruct the visualization engine to perform interference detection computations in addition to performing the regular core VITASCOPE computations that define and manipulate objects on virtual construction sites. In addition, the add-on defines several new language statements that provide users with precise control over (1) which pairs of scene objects are monitored and tested for potential collisions, and (2) the nature and semantics of the feedback that is generated when interferences among various scene objects are actually detected.

C-COLLIDE also defines special statements to describe arbitrarily shaped abstract construction resources (e.g., hazard or protected spaces) in process visualizations. Such resources need not necessarily have a physical representation (i.e., they might be invisible) but are nevertheless relevant in the context of interference analysis (e.g., hazard or protected space incursions). C-COLLIDE's parametric text statements can be used together in several ways to detect, analyze, and report any occurring interferences in construction process visualizations.

Table 1 presents the animation statements C-COLLIDE implements and briefly indicates their usage. In case of redefined (i.e., supplemented) statements that are part of the core VITASCOPE language, the last column indicates their original usage. Statements containing a N/A entry in the last column are original C-COLLIDE statements that have not been previously defined in VITASCOPE. In cases of statements that are redefined, column two indicates the functionality that C-COLLIDE appends to the original statements. For original C-COLLIDE statements, column two describes the complete statement functionality.

5.1 Object definition and initialization

C-COLLIDE redefines several core VITASCOPE statements in order to perform the supplemental computation necessary to initialize the interference detection routines as objects are instantiated and introduced in scenes. The supplemental computation involves converting scene object representations into formats that the interference detection routines can read and operate on. C-COLLIDE's algorithms require each instantiated scene object (represented by a 3D CAD model) to be an arbitrarily large set of

Table 1 Usage of C-COLLIDE statements

Statement	C-Collide appended/defined usage	Original usage
CLASS [ClassName] [CADFileName];	Converts the input CAD file into a format interpretable by the interference detection algorithm	Associates a class of simulation entities with their geometric description contained in a CAD file
CREATE [ObjName] [ClassName];	Constructs, associates, and stores the OBBTrees of the created simulation objects	Creates specific simulation objects by instantiating predefined classes
PLACE[ObjName] [AT/ON] [Location];	Notifies the collision detection engine about the presence of new objects. This includes objects attached to the current object	Places simulation objects at particular locations or at the beginning of resource movement paths
ATTACH [ChildObjName] [ParentObjName] [AttachPoint];	Notifies the collision detection engine about the presence of new scene objects if the parent objects are already in the scene	Attaches objects to one another at a specified pivot point
ACTIVATEOBJECT [ObjName];	Turns on collision tests for all object pairs involving this object	N/A
DEACTIVATEOBJECT [ObjName];	Turns off collision tests for all object pairs involving this object	N/A
ACTIVATEPAIR [ObjName1] [ObjName2];	Turns on collision tests for a specific pair of objects. Both objects must be active	N/A
DEACTIVATEPAIR [ObjName1] [ObjName2];	Turns off collision tests for a specific pair of objects	N/A
ABSTRACTOBJECT [ObjName] [ClassName];	Creates an arbitrarily shaped abstract (i.e., invisible) scene object	N/A
RESPONSEMODE [INTERACTIVE/SILENT] [LogFileName];	Indicates whether C-COLLIDE must interactively report collisions or record them silently to a disk log file	N/A

disconnected primitive triangles. These triangles must together describe all the surface and internal features of a scene object.

CAD models used by VITASCOPE to describe the geometry of simulation objects can be imported from a wide variety of sources in different file formats. These CAD models can be internally described as a set of disconnected triangles but can also be defined in a wide variety of other CAD model representations such as parametric surfaces, constructive solid geometry, implicit surfaces, and polygon sets with topological information. In order to detect interferences between simulation objects then, C-COLLIDE must first convert all input CAD models into the triangulated format that it can read and operate on.

C-COLLIDE provides an internal CAD model converter that triangulates (if necessary) input CAD files into a format suitable for the adopted interference detection algorithm. This conversion is performed each time a CLASS statement associates a group of simulation entities with their geometric description by supplying a 3D CAD file. While the core VITASCOPE functionality of instantiating, manipulating, and displaying objects in a scene is accomplished using the original input CAD model, C-COLLIDE uses the converted model representation in all its computations. Figure 8 juxtaposes an original CAD representation of a concrete truck against its converted triangulated representation. VITASCOPE manipulates and displays the original model during visualization. However, C-COLLIDE computations that test for interferences involving any virtual concrete trucks are performed using the triangulated representation. This triangulated representation is obviously not displayed in the virtual world during visualization and is meant for internal computation only.

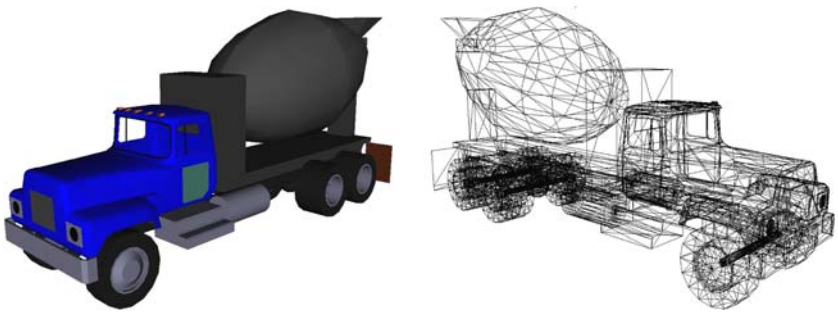

Fig. 8 Original and triangulated CAD model representations

The other core statements (CREATE, PLACE, and ATTACH) that C-COLLIDE redefines in effect build, monitor, and maintain an additional, parallel, invisible scene database that is a copy (albeit in a different format) of the scene database that VITASCOPE constructs and manages to display virtual construction processes. While VITASCOPE uses the original scene database for scene manipulation and rendering, C-COLLIDE performs the interference detection computations using its internal scene database copy. Obviously, C-COLLIDE'S database must be updated each time VITASCOPE manipulates the scene.

5.2 Interference detection control

C-COLLIDE's interference detection control statements allow users to explicitly and precisely specify contextual rules and assumptions to the collision detection engine. As discussed earlier, this capability can be used prudently to make the interference detection calculations more efficient by eliminating from the computation pipeline pairs of scene objects that are not likely to interact. Examples of such objects include pieces of equipment working on different areas of a jobsite or trades working on different floors of a building under construction.

C-COLLIDE provides two components of statements for interference detection control and contextual optimization. These statements allow users to specify which objects should be monitored and tested for collisions. The first per-object component defines statements (ACTIVATEOBJECT, DEACTIVATEOBJECT) that operate on a single scene object and either add or remove from the computation pipeline all possible pairs of scene objects containing the object operated upon. The second pair-wise component defines statements (ACTIVATEPAIR, DEACTIVATEPAIR) that operate on a specific pair of scene objects either turning interference detection among them on or off.

All created C-COLLIDE objects are active by default. In addition, all pairs of scene objects that include the newly created object are also activated initially. C-COLLIDE manages these two control components separately. Thus, for a pair of objects to be tested for intersection or collision, not only must the pair be active, but each of the two objects must be active as well. The following section elucidates the usage of these control statements.

5.3 Validation

Figure 9 presents an animation snapshot of a bridge construction site. On this job, fresh concrete manufactured at a batch plant (visible on the far left) on the shore was placed in hollow steel jackets to cast the piers of the bridge. Concrete was delivered to the workface at the piers using barges to transport concrete trucks. Concrete from each arriving truck was pumped into a hopper built on the pier's working platform. The hopper fed a tremie pipe that was lowered into each steel jacket. As the depth of placed concrete rose, a crane mounted on a floating platform withdrew sections of the tremie pipe from the jacket and placed them on racks after being cut. This procedure continued until the entire depth of the jacket was concreted.

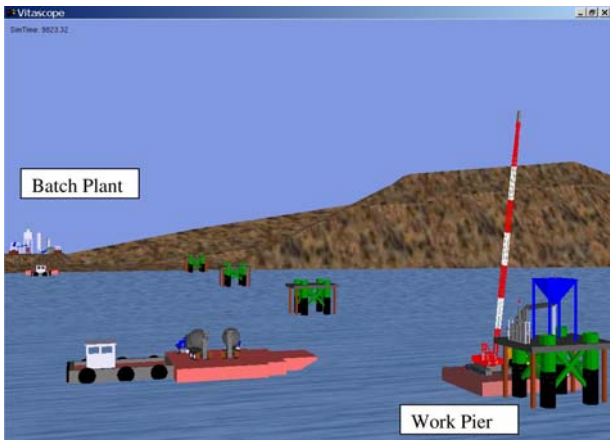

Fig. 9. Animation snapshot of a bridge construction jobsite

Figure 10 presents a portion of the animation trace file that uses C-COLLIDE's control statements to specify simple contextual rules and assumptions to the collision detection engine. Of particular interest is the scene object JobSite that describes the terrain (including the water surface) and the landscape of the jobsite. Most scene objects (i.e., barges, trucks, floating platforms, batch plant, etc.) are obviously in contact with the surface at all times. Other scene objects such as the crane and the concrete pump operate on the floating platform and are not anticipated to be in contact with the surface for the duration of the animated processes.

Since both the terrain and the other scene objects are represented by CAD models, the perennial contact between them is geometric interference from the collision detection algorithm's viewpoint. From the user's perspective, however, such interaction (e.g., barge hulls touching the water, truck tires touching the terrain) is obviously permissible. The fifth statement thus turns off collision among all pairs of scene objects involving the JobSite object. In addition, the crane mounted on the floating platform is not anticipated to be in close proximity to the batch plant during the duration of the

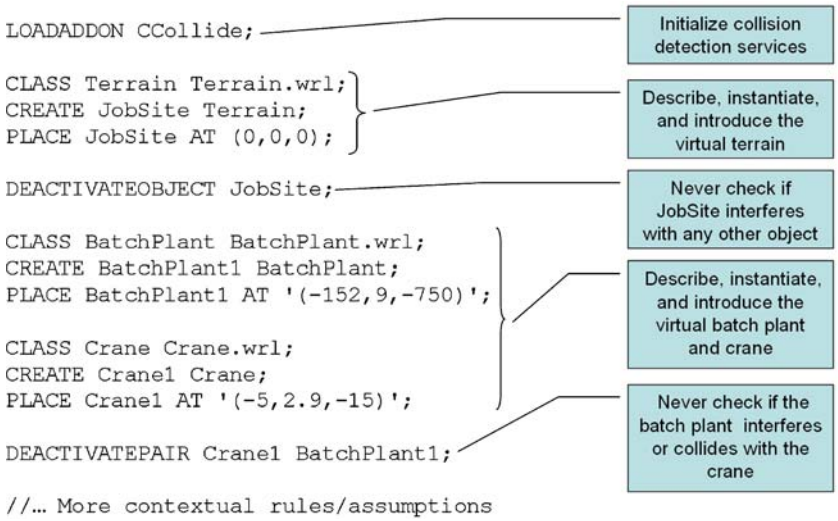

Fig. 10 Specification of contextual rules and assumptions

animated operations. The final statement conveys this assumption to the collision detection engine. Such contextual assumptions dramatically reduce the number of scene object pairs that C-COLLIDE must monitor for possible collisions increasing the algorithm's computational efficiency manifold.

In this particular animation, the maximum number of scene object pairs that were present on the simulated site and were being continuously monitored for possible interference with each other is about 112. The complexity and the number of simulation entities present in the described example are typical of most simulated construction operations. On a standard laptop computer with 512 MB RAM and a 2.4 GHz Pentium 4 processor with an NVidia Quadro graphics card, a sustained screen refresh rate in excess of 24 frames/s was observed. Similar results were obtained with other animations in which C-COLLIDE was tested.

5.4 Collision feedback and response

C-COLLIDE tests for possible interference between all active pairs of scene objects at every time instant (i.e., frame). C-COLLIDE defines a statement (RESPONSEMODE) that allows users to specify the nature of the feedback that must be generated when collisions between active scene object pairs are detected. Two response modes—interactive and silent—are currently defined. In interactive mode, C-COLLIDE pauses a running animation and outputs details about each detected interference. C-COLLIDE then waits for the user to specify whether (1) the interference detected should be ignored, (2) the animation should be aborted, or (3) the scene object pair should be deactivated from further computations.

In silent mode, C-COLLIDE does not interrupt a running animation even if interferences are detected. Instead, details about all collisions that occur during an animation run are time stamped and written to a formatted disk file. Users can later analyze the outputted interference log for any collisions that might have occurred during visualization. C-COLLIDE's silent collision response mode is particularly useful for detecting interferences in long animations that span hours or even days and for detecting collisions that occur rarely (i.e., rare simulation events). In both cases, interactive detection would require hours of visualization even when using a high viewing ratio (ratio of simulation time to wall clock time).

6 Advantages of add-on approach

Collision detection and interference analysis is considered to be the most challenging problem in visual simulations. The collision detection algorithms adopted by any visual simulation application usually influence the core components involved in the computation loop. This is evident from the fact that C-COLLDE redefines (i.e., augments) several core VITASCOPE statements in order to implement collision detection capabilities. VITASCOPE's ability to allow such core computation components to be implemented as add-ons to the main visualization engine has several distinct advantages.

In C-COLLIDE's case, the adopted add-on approach allows us to modify any portion of the collision detection routines without having to modify the

implementation of VITASCOPE's main visualization engine. In addition, this approach provides the flexibility that allows any researcher to implement, if necessary, an entirely different collision detection algorithm for use in VITASCOPE visualizations. This is of significant importance and relevance since fundamental work on efficient collision detection algorithms is still an active research area in computer graphics. By implementing its collision detection algorithms as an add-on, C-COLLIDE provides users the option of easily replacing VITASCOPE's interference detection capabilities with any other algorithms of their choice.

7 Conclusions and future work

The presented research capitalizes on advanced 3D geometric collision detection algorithms to design efficient mechanisms for interference detection, control, and response in dynamic 3D construction process visualizations. The methods defined by C-COLLIDE allow users to identify all undesirable conflicts that can occur among static, dynamic, and abstract construction resources in construction process visualizations.

C-COLLIDE's interference detection algorithms only detect the occurrence of ''hard'' interferences between physical scene components and report the exact point(s) of contact if queried. C-COLLIDE's algorithms provide no routines to compute and report distances between two scene objects. Such routines can be useful in construction process visualizations as they can allow the detection of ''soft'' interferences such as minimum clearance violations between any pair of stationary or mobile scene objects. C-COLLIDE does provide a statement (ABSTRACTOBJECT), however, that can be utilized as a workaround to define an invisible, protected envelope around any scene object.

Since research on efficient collision detection algorithms is ongoing in computer graphics, the design of any new algorithms or techniques presents an opportunity to improve upon or replace the collision detection algorithms adopted in C-COLLIDE. Of course, any such improved algorithms must be able to conform to VITASCOPE's requirements and its add-on interface. From a VITASCOPE user's perspective, C-COLLIDE provides users with a comprehensive test bed for several studies—such as process or craft level spatial conflict analyses or space usage analyses—that require the capability to detect interferences among virtual resources in dynamic construction process visualizations.

Acknowledgments The authors gratefully acknowledge the support of the National Science Foundation (NSF) CAREER (Grant CMS-9733267) and ITR (Grant CMS-0113890) programs. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF.

References

1. Akinci B, Fischer M, Levitt R, Carlson R (2002) Formalization and automation of time–space conflict analysis. J Comput Civil Eng 16(2):124–134
2. Bentley Systems Inc. (2000) Bentley dynamic animator user guide, Exton, PA
3. Bentley Systems Inc. (1999) Bentley interference manager user guide, Exton, PA
4. Cohen JD, Lin MC, Manocha D, Ponamgi M (1995) I-COLLIDE: an interactive and exact collision detection system for large-scaled environments. In: Proceedings of the ACM international symposium on interactive 3D graphics. Association for Computing Machinery, New York, pp 189–196
5. Dias JMS, Gamito M (2001) Automatic design verification in building construction. In: Proceedings of the specialty conference on fully integrated and automated project processes. ASCE, Reston, VA, pp 81–95
6. Gottschalk S, Lin MC, Manocha D (1996) OBB-Tree: a hierarchical structure for rapid interference detection. In: Proceedings of the 23rd annual conference on computer graphics and interactive techniques, Association for Computing Machinery, New York, pp 171–180
7. Hudson TC, Lin MC, Cohen J, Gottschalk S, Manocha D (1997) V-COLLIDE: accelerated collision detection for VRML. In: Proceedings of VRML 97, for Computing Machinery, New York, pp 117–123
8. Kamat VR (2003) VITASCOPE: extensible and scalable 3D visualization of simulated construction operations, PhD Dissertation, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg
9. Kamat VR, Martinez JC (2001) Visualizing simulated construction operations in 3D. J Comput Civil Eng 15(4):329–337
10. Kamat VR, Martinez JC (2002) Scene graph and frame update algorithms for smooth and scalable 3D visualization of simulated construction operations. J Comput Aided Civil Infrastruct Eng 17(4):228–245
11. Kamat VR, Martinez JC (2004) Dynamic three-dimensional visualization of fluid construction materials. J Comput Civil Eng 18(3):237–247
12. Kamat VR, Martinez JC (2005) Dynamic 3D visualization of articulated construction equipment. J Comput Civil Eng 19(4):356–368
13. Law AM, Kelton WD (2000) Simulation modeling and analysis, 3rd edn. McGraw-Hill, New York
14. Lin M, Gottschalk S (1998) Collision detection between geometric models: a survey. In: Proceedings of the IMA conference on mathematics of surfaces. Available at: (http://www.cs.unc.edu/~dm /collision.html)
15. Ponamgi MK, Manocha D, Lin MC (1997) Incremental algorithms for collision detection between solid models. IEEE Trans Vis Comput Graph 3(1):51–64