Efficient Interference Detection in 3D Animations of Simulated Construction Operations

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

1Assistant Professor, Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI – 48109, USA; PH (734) 764-4325; FAX (734) 764-4292; email: vkamat@umich.edu
2Associate Professor, Department of Civil and Environmental Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA – 24061, USA; PH (540) 231-9420; FAX (540) 231-7532; email: julio@vt.edu

Abstract

This paper presents research that led to the design and implementation of efficient interference 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 discrete-event simulation models. The designed methods are implemented in a software tool called C-Collide that integrates as an add-on with the VITASCOPE visualization system.

Common types of clashes that can occur on real construction sites and that C-Collide can identify beforehand in 3D process animations 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).

Animations of simulated construction processes are computationally demanding because elapsed visualization time is always related to the elapsed wall clock time by a distinct viewing ratio. This viewing ratio must be 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, the screen must be updated at an acceptable refresh rate (ideally >= 24 fps). The impact of the collision detection computations on this frame rate must thus be minimal.

Computational efficiency in the designed interference detection methods was therefore of paramount importance in C-Collide. In addition, this efficiency could not be traded off against accuracy. In order to achieve this, the authors capitalized on advanced documented algorithms for efficient collision detection between arbitrarily moving 3D geometric objects to design mechanisms for interference detection, control, and response in 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 a lucid understanding of all object motions and potential interferences in any area of activity on a simulated construction job site.
**Introduction**

In this paper, we present a tool, C-COLLIDE, that can be used to 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. We capitalize on advanced documented algorithms for efficient collision detection between arbitrarily moving 3D geometric objects to design mechanisms for interference detection, control, and response in 3D construction process visualizations. C-COLLIDE is implemented as an add-on (extension) to the VITASCOPE visualization system. VITASCOPE is a user-extensible 3D animation language designed for visualizing modeled construction operations in smooth, dynamic 3D virtual worlds (Kamat and Martinez 2004). C-COLLIDE’s interference detection capabilities dynamically check each motion of VITASCOPE scene objects to determine if any pairs of scene objects interfere. This provides a lucid understanding of all object motions and potential interferences in any area of activity on a simulated construction job site.

Previous research in construction has applied collision detection algorithms in various static and dynamic graphical visualization contexts. Applications such as Interference Manager (Bentley 1999) and the Interference Detection Language (IDL) (Dias and Gamito 2001) use collision detection algorithms to identify geometric clashes between permanent facility components and automatically verify the design of a proposed facility. Akinci et al. (2002) 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 2000) utilize collision detection algorithms to check for geometric interferences between objects during playback of manually created motion sequences.

Neither of these applications, however, is real-time. In other words, there is no relation between elapsed visualization 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 real-time because elapsed visualization time is always related to the elapsed wall clock time by a distinct viewing ratio. 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 per second). The impact of any involved collision detection computations on this frame rate must be minimal. C-COLLIDE’s main contributions are:

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

The collision detection problem in a dynamic 3D virtual world consisting of many moving and stationary objects can be generally separated into three distinct steps (Lin and Gottschalk 1998):

- 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

Relevant Taxonomy. The algorithms to accomplish the first two 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 – These algorithms 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 (1998) for a comprehensive and informative survey on these taxonomies and other discussion.

Technical Approach

We have adopted a two-phase, hierarchical multi-level approach presented in Hudson et al. (1997) 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 our purposes because during each intersection test (one per frame), we can quickly eliminate pairs of scene objects that are not within interacting distance of each other. 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. Interference Detection Algorithm Architecture.

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 (Cohen et al. 1995). 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 (Gottschalk et al. 1996). 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.

**Pruning Scene Object Pairs.** In the sweep and prune phase, the 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. AABBs describe loosely fitting bounding volumes for all but the most ideal enclosed object orientation. Figure 2 illustrates this graphically by projecting in two dimensions.

**Figure 2. Axis Aligned Bounding Boxes.**
The constructed bounding boxes are analyzed in 3D space to determine overlapping pairs. A dimension reduction approach is used for the purpose (Cohen et al. 1995). The sweep and prune algorithm begins by projecting each three-dimensional bounding box onto the three coordinate axes. Since the bounding boxes are axis aligned, 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 figure 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 objects that pass the sweep and prune test are passed to the detailed collision detection phase.

**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.
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 figure 5, has two components. First is the placement of a tight fitting OBB around each (sub)object (represented as a collection of 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 (Gottschalk et al. 1996). 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.
**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.

Starting 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”

*Figure 6. Traversing Bounding Volume Hierarchies to Detect Collisions.*

If pairs of leaf OBBs are found to be intersecting during the OBBTree traversal, the algorithm performs deterministic 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 in an order of increasing accuracy (and computation cost). By adopting this hierarchical, multi-level procedure, the algorithm is thus able to provide us with exact collision detection capabilities while maximizing computational efficiency.

**C-Collide**

We have implemented the algorithms described above in a tool called C-COLLIDE, which integrates as an extension (add-on) with the VITASCOPE visualization system. 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 the regular core computations that define and manipulate objects on virtual construction sites. In addition, the add-on defines several new language statements that provide 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. Table 1 presents the statements that C-COLLIDE implements. In case of redefined VITASCOPE statements, the last column indicates their original usage. Statements containing a N/A entry are original C-COLLIDE statements. In cases of redefined statements, column two indicates the functionality that C-COLLIDE appends to the original statements. For original C-COLLIDE statements, column two describes their complete functionality.
Table 1. Usage of C-COLLIDE Statements.

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

Object Definition and Initialization. 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. In order to detect interferences between simulation objects, C-COLLIDE must first convert all CAD models into a triangulated format that it can operate on. This conversion is performed each time a CLASS statement is processed. 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.
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 (in a different format) of the scene database that VITASCOPE constructs and manages to display virtual construction processes. While VITASCOPE uses the original database for scene manipulation and rendering, C-COLLIDE performs interference detection computations using its internal database copy.

**Interference Detection Control.** Figure 9 presents an animation snapshot of a bridge construction site. On this job, 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.
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 engineer’s perspective, however, such interaction (e.g. barges 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 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.

**Collision Feedback and Response.** C-COLLIDE defines a statement (RESPONSEMODE) that allows engineers 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. Engineers can later analyze the 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).

**Conclusions and Future Work**

The presented work 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 mechanisms defined by C-COLLIDE allow engineers to identify any and all undesirable conflicts that can occur among static, dynamic, and abstract construction resources in simulated 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 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 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**


Collision Detection System for Large-Scaled Environments”, Proceedings of the ACM
International Symposium on Interactive 3D Graphics, Association for Computing Machinery,
New York, NY, 189-196.
Proceedings of the Specialty Conference on Fully Integrated and Automated Project Processes,
ASCE, Reston, VA, 81-95.
Interference Detection”, Proceedings of the 23rd annual conference on Computer Graphics and
Interactive Techniques, Association for Computing Machinery, New York, NY, 171-180.
Accelerated Collision Detection for VRML”, Proceedings of VRML 97, for Computing
Machinery, New York, NY, 117-123.
Society of Civil Engineers, Reston, VA, 237-247.
Proceedings of the IMA Conference on Mathematics of Surfaces, Available:
(http://www.cs.unc.edu/~dm/collision.html).
York, NY.
Between Solid Models”, IEEE Transactions on Visualization and Computer Graphics, Vol. 3,
No. 1, IEEE, Piscataway, NJ, 51-64.