Determining Collisions between Moving Spheres for Distributed Virtual Environments

Kier Storey, Fengyun Lu, Graham Morgan
School of Computing Science
Newcastle University, Newcastle upon Tyne, NE1 7RU, UK
Telephone: + 44 191 222 7983, Fax: + 44 191 222 8232
E-mail: {Kier.Storey, Fengyun.Lu, Graham.Morgan}@newcastle.ac.uk

Abstract

We present an approach to collision detection that is appropriate for satisfying the requirements of interest management schemes used in distributed virtual environments. Such environments are characterized by their distributed deployment over a number of nodes connected via a computer network. The aim of an interest management scheme is to identify when objects that populate a simulation supported by a distributed virtual environment (objects could be hosted on different nodes) should be interacting via message exchange while preventing objects that should not be interacting from exchanging messages. The approach to collision detection presented in this paper produces accurate results when determining object interactions. Furthermore, we present variations on our approach that exploit any coherence that may exist in a simulation to provide a solution that may scale for large numbers of objects.

1. Introduction

Applications that simulate some geographical model populated with moving objects require collision detection algorithms to identify when objects collide. A number of these applications allow a user to interact with a simulation, and possibly other users, in real-time. Such applications have been used for training purposes [2], computer supported collaborative work (CSCW) [1] [4] and social play [3]. These applications require collisions to be identified between all objects in a timely fashion. Satisfying this requirement ensures there exists no collisions that go undetected by the application. Simulations may contain thousands of objects, making the delivery of real-time collision detection a challenging problem. A further complication introduced by many applications is the inability to predict accurately the position and orientation of objects in advance due to the non-predictive nature of object movements, ruling out the ability to solve collision detection as a function of time.

Collision detection is an important requirement when applications present a simulation of moving solid objects. Assuming the simulation is an approximation of some real-world location, the ability to view objects as solid is essential for promoting the validity of the simulation. Solid objects should not share the same area of the virtual world. In this scenario, collision detection and response algorithms present a more realistic simulation to users. However, the modelling of solid objects may not be the only time an application may require a collision detection algorithm.

Distributed Virtual Environments (DVEs) provide a geographical representation of a virtual world that may be navigated by geographically dispersed users. Users may interact with each other and the virtual world in real-time. Objects that populate a simulation supported by a DVE may reside on different nodes, requiring objects to participate in message exchange (possibly across computer networks such as the Internet) in order to ensure objects are aware of each other's actions (e.g., vector position updates as a result of object movement). Propagating actions to all objects is not scalable due to the processing overheads associated to message exchange and delays imposed on message delivery by the underlying network. Therefore, limiting the number of recipient objects associated to a message is desirable. This may be achieved by identifying the geographic area of a virtual world an object may exert influence and limiting the sending of messages to only those recipient objects that exist in the sender object's area of influence. This requires an approach to collision detection that determines which objects (likely recipients) are present in a sender object's area of influence. Unlike the modelling of solid objects, objects are expected to exist within an area of influence for some time. An approach to collision detection is required that returns a set of objects identifying the recipients for a sender object's message based on their shared occupation of the sender object's area of influence.

In this paper we present an approach to collision detection which is well suited to satisfying the requirements of a DVE. We attempt to satisfy real-time requirements of a DVE by reducing the number of comparisons needed to derive the sets of objects identified as the recipient of a message.

The rest of the paper is organized as follows. In section 2 we describe background and related work. In section 3 we describe our approach to collision detection. In section 4 we discuss the performance of our approach and in section 5 we present our conclusions.

2. Background and Related Work

Algorithms that compare all objects with all other objects implement a brute force approach to collision detection. Such an approach is an $O(n^2)$ problem and satisfying collision detection requirements for large numbers of objects may not be possible in real-time. This problem has been well studied in the literature and a number of algorithms have been proposed that perform better than $O(n^2)$ [5] [6] [7] [8] [9].

Temporal coherence that exists in a simulation has been exploited when determining collisions [8] [9]. An application may exploit temporal coherence if the state of a simulation does not change significantly between consecutive state updates. For the purposes of collision detection, this equates to objects moving only slightly from one frame of animation to the next. Collision detection based on axis aligned bounding boxes (AABBs) using the sweep and prune approach is a well known technique for exploiting coherence [9]. This is achieved via the projection of the bounding box onto the x, y and z axes with the sorting of the endpoints allowing overlap to be detected separately on each axis. If an AABB, say A_1, has end points that come after a different AABB's, say A_2's, start points but before A_2's end points on the x, y and z axes then A_2 collides with A_1. As the assumption is taken that objects travel relatively small distances between frames, coherence is exploited via the fact that the points associated to AABBs may be sorted in near linear time on each axis as the distribution of points associated to AABBs along each axis are already nearly sorted from the previous iteration of the algorithm.

Spheres are commonly used in computer graphics for modelling objects and exploiting coherence to reduce the number of pairwise comparisons when determining sphere collisions [10] [11]. In this paper we are primarily interested in applying sphere based collision detection techniques to aid in deriving scalable DVEs.

We assume a DVE represents a geographic (virtual) space containing objects that may navigate such a space. The DVE is deployed across geographically separated nodes connected by an underlying network. Each node may host a number of objects, their local objects, with nodes responsible for informing each other of the actions (e.g., movement) of local objects via the exchange of messages across the network. Limiting network traffic and unnecessary message processing is desirable to achieve scalability. This may be possible via the division of the virtual space, only allowing a node, say N_1, to exchange messages with another node, say N_2, when one or more objects hosted by N_1 coexist in the same division of virtual space as one or more objects hosted by N_2. *Interest management* is the term commonly used to describe restricted message dissemination between objects using virtual space division. Interest management may be classified into two categories: (i) region based [2]; (ii) aura based [12]. In the region based approach the virtual world is commonly, but not always, divided into well defined uniform sized regions that are static in nature (i.e., their boundaries are defined at virtual world creation time). The recipient of a message is limited to only interested participants (i.e., reside within the same, or neighbouring, region as the sender). In the aura based approach each object is associated to an aura that defines an area of the virtual world over which an object may exert influence. Ideally, an object may potentially communicate their actions to only objects that fall within their influence. An aura is commonly defined as a sphere.

The aura approach provides a basis for accurately modelling the degree of influence between objects [13] and provides a model for appropriate implementation of an interest management scheme. However, in practise the regionalisation approach tends to be favoured as spatial sub-division collision detection algorithms are well suited to their needs [5] [14] [15]. Successful implementation of the aura based approach requires an approach to collision detection that is based on spheres and also produces appropriate sets that indicate which objects should be the recipient of a message.

In this paper we present an approach to collision detection that is suited to aura based interest management schemes. Our approach reduces the number of pairwise comparisons that brute force would result in and provides appropriate sets indicating potential recipients of a message. Furthermore, we reduce the pairwise comparisons still further by exploiting coherence that exists in the simulation. We describe two variations of our approach in which we exploit coherence. Both of our variations exploit coherence for improved performance with one of our variations based on the well known sweep and prune approach popular with AABB based collision detection algorithms. Initial performance results look promising.

3. Collision Detection for DVEs

In this section we describe our approach to collision detection. We explain how the properties of spheres may be exploited to derive sets of intersecting spheres (spheres that partially or fully share an area of the virtual world) in fewer comparisons than would be expected when using a brute force approach. We then describe two enhancements that use coherence to speed up the ability to derive sets of intersecting spheres and reduce the number of comparison tests required to gain such sets.

3.1 Expanding Sphere

We assume each object present in the virtual world to be a sphere (for clarity we refer to spheres/auras as objects from now on) with each object in the virtual world a member of the set *VR*. A collision between two objects, say O_a and O_b, is said to have occurred if O_a and O_b intersect (i.e., there exists an area of the virtual world that lies within the spheres of O_a and O_b). A *collision relation* (*CR*) identifies a set of objects that share, in part or fully, an area of the virtual world. Therefore, an object that belongs to a collision relation, say CR_i, collides with every other object that belongs to CR_i. A collision relation may contain one or more objects. A collision relation that contains only a single object indicates that such an object does not collide with any other object in the virtual world. These collision relations provide the object groups that dictate the appropriate recipients of a message for an interest management implementation. A set *SU* contains objects that are to be considered for collision; hence *SU* contains a subset of objects in the virtual world such that $SU = \{O_1, O_2, O_3, ..., O_n\}$. The determination of this subset is arbitrary, but will usually be equivalent to the full membership of *VR* at the start of the process of determining collisions. When an object, say O_a, from *SU* is considered for collision it is said to be the *object under consideration* (*OUC* – when an object, say O_a, becomes the object under consideration we write O_a^{OUC}). A set *SC* contains the collision relations between objects previously identified as the *OUC*. A collision relation is an element of set *SC*. For example, if $SC = \{\{O_a, O_b, O_c, O_d\}, \{O_a, O_e\}, \{O_f, O_g\}, \{O_h\}\}$ we may state that *SC* contains four collision relations that are made up from the objects O_a, O_b, O_c, O_d, O_e, O_f, O_g and O_h. A 2D graphical representation of the collisions between these objects is shown in figure 1 (the scheme is applicable in 3D virtual worlds but we limit our diagrams to 2D for clarity).

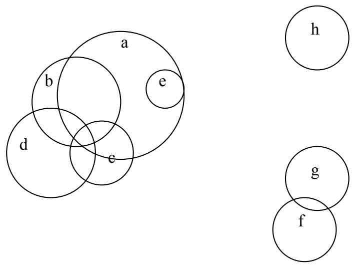


Figure 1 – Collision relations.

We now consider the identification of collision relations in more detail. We base our approach on determining if two spheres collide, say O_a and O_b, if the distance separating O_a and O_b is less than the sum of the radii of O_a and O_b. We assume an object's position vector and radius is known and may be described as O^{pos} and O^{rad} respectively. Additionally, our method also requires a *CR* to maintain position vector and radius information (CR^{pos}, CR^{rad}). CR^{pos} is taken from the first object identified in the *CR* and CR^{rad} is also initially taken from the first object identified in the *CR* when the *CR* is created (i.e., *CR* only has one object within it). CR^{rad} is re-evaluated each time an object is added to a *CR*. Assume an object, say O_x, is to be added to a *CR*, say CR_i, the value of CR_i^{rad} is incremented using the following method:

1. Let *SD* be the separating distance between O_x^{pos} and CR_i^{pos}.
2. If $SD+O_x^{rad}$ is less than or equal to CR_i^{rad} then CR_i^{rad} remains unchanged.
3. If $SD+O_x^{rad}$ is greater than CR_i^{rad} then let CR_i^{rad} become $SD+O_x^{rad}$.

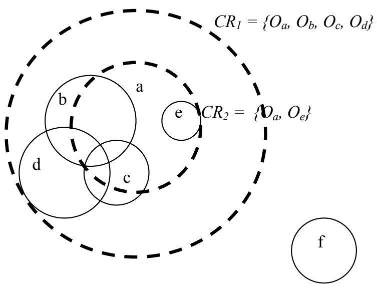


Figure 2 – Minimising comparisons via collision relations.

When CR^{rad} remains unchanged after the addition of an object (as in 2 above) the new object lies fully within CR^{rad} (this would be the case when adding O_e to the collision relation that contains O_a to make *CR* =

$\{O_a, O_e\}$ in the example shown in figure 1). When *CR^{rad}* is changed (as in 3 above) the new object is not fully within *CR^{rad}* and therefore we must extend *CR^{rad}* to encompass the new object (this would be the case when adding *O_d* to the collision relation that contains *O_a*, *O_b* and *O_c* to make *$CR = \{O_a, O_b, O_c, O_d\}$* in the example shown in figure 1). Extending *CR^{rad}* in this manner provides an opportunity to reduce the number of comparisons that need to be made in determining which objects intersect. In figure 2 we show how this can be possible by further considering the example shown in figure 1 when determining the collision relations of *O_f* (*O_f* is the *OUC*). Assume we have already deduced collision relations and calculated their associated *CR^{rad}* values for the first five objects considered in alphabetical order. The dotted circles in the diagram identify the expanded radii representing *CR_1* and *CR_2* (*CR_2* is actually the true radius of *O_a*). It is clear that *O_f* does not intersect *CR_1* and *CR_2*. Therefore *O_f* cannot intersect with any of the objects that lie within *CR_1* and *CR_2*. By comparing *O_f* with *CR_1* and *CR_2* (two comparisons), we do not need to compare *O_f* with any of the objects within *CR_1* and *CR_2* (five comparisons).

Pseudo code main collision detection algorithm:

```
Algorithm CollisionDetection
Inputs  SU : Set of Objects;
Returns SC : Set of CRs // CR - Collision Relation;
Variables  $O_x$, $O_i$: Object; CL, $CR_y$: CRs; newCRs: Set of CRs

Begin
 SC:= $\varnothing$;
 for each $O_x \in$ SU do
  newCRs:= $\varnothing$;
  for each $CR_y \in$ SC do
   if ($O_x$ collides with $CR_y$) then
    CL:= $\varnothing$;
    /** Find colliding objects in CR **/
    for each $O_i \in CR_y$ do
     if ($O_x$ collides with $O_i$) then CL:=CL $\cup$ {$O_i$} fi
    od
    /** Remove CRy if complete collision, add later **/
    if (card CL = card $CR_y$) then SC:=SC\{$CR_y$} fi
    if (CL $\neq \varnothing$) then
     /** Turn CL into a CR by adding $O_x$ **/
     CL:=CL $\cup$ {$O_x$};
     /** Add to set of new CRs for $O_x$ **/
     /** Check for sub/super sets in newCRs **/
     newCRs:= addCR(CL, newCRs);
    fi
   fi
  od
  /** Add new CRs for $O_x$ to SC **/
  if (newCRs = $\varnothing$) then /* Add singleton CR */
   SC:= SC $\cup$ {{$O_x$}};
  else
   SC:= SC $\cup$ newCRs;
  fi
 od
 return SC;
End
```

Figure 3 – Pseudo code describing algorithm.

We now describe the algorithm required for determining all collision relations from the set of objects contained within *VR*. *SU* is initialised to be the equivalent of *VR*. Each element in *SU* is considered in turn and identified as the *OUC*. The *OUC* is compared with each *CR* in *SC* (an element of the set *SC* is a *CR* which has fully or partially been deduced during an iteration of the algorithm). If the *OUC* has not collided with any *CR*s then the *OUC* is appended to *SC* as a new *CR*. If *SC* is empty (i.e., this is the first member of *SU* to be identified as the *OUC*) then the *OUC* is immediately placed in *SC*. When a collision between the *OUC* and a *CR*, say *CR_i*, is identified then each member object of *CR_i* is compared to the *OUC* to identify *OUC*/object collisions. If the *OUC* collides with every object in *CR_i* then *CR_i^{rad}* is updated and the *OUC* is appended to *CR_i*. If the *OUC* does not collide with any objects within *CR_i* then *CR_i* remains unchanged. If the *OUC* collides with a strict subset of objects in *CR_i* then this is potentially a new collision relation and is placed in a temporary set *collision list* (*CL*). Each time the *OUC* is compared to a different *CR* for collision purposes the *CL* is initialised to empty set. Therefore, after the *OUC* has been compared with all existing objects within a *CR* then a *CL* may contain a new *CR* to be appended to *SC*. Duplicate entries are not allowed in *CL* (this prevents duplicate *CR*s from been appended to *SC*). The pseudo code representing the algorithm is presented in figure 3. For clarity, the duplication test is not shown as it is located in the function `addCR(CL, newCR)`.

3.2 Ordered Collision Relations

In the expanding sphere algorithm just described we consider the order in which members of the set *SU* become the *OUC* to be arbitrary. We now impose ordering on the set *SU* to derive a sequence (*SU^**) for dictating the order in which members of *SU* become the *OUC* in an effort to take advantage of coherence. Ideally, we require *CR*s with the largest cardinality to be identified early in the execution of the algorithm to ensure the least number of comparisons to derive *SC*. This happens in the example in figure 2, where the *CR*s are formed in the order of their cardinality (largest to smallest) when considering objects in alphabetical order. We may show the benefit of early identification of *CR*s with the largest cardinality by considering a new ordering of *OUC* identification in figure 2. As *O_h* is considered the *OUC* last then three comparisons are required to rule out the need to compare *O_h* with all the other objects (7 comparisons). However, if we change the ordering only slightly and assume *O_h* is considered the *OUC* first then all other objects will be compared with O_h as they assume the role of the *OUC* (7 comparisons).

The ordering of when members of SU^* become the *OUC* is based on the cardinality of the *CR*s derived from the previous iteration of the expanding sphere algorithm. For this purpose the set *SC* is ordered based on the cardinality of *CR*s from largest to smallest to form SC^*. Once ordered, each *CR* in SC^* is considered in turn, with the object membership of a *CR* added to SU^*. As an object may occur in multiple *CR*s, there is a need to ensure that if an object already exists in SU^* (previously added) it is not added again. This guarantees that an object does not become the *OUC* more than once during the same iteration of the algorithm (this could result in repeat comparisons).

The first iteration of the algorithm in any given run will have to rely on *SU* and not the ordered sequence SU^* to dictate the order in which objects become the *OUC* (as there is no SC^* from a previous iteration of the algorithm on which to base the ordering of elements in SU^*). However, after the completion of the first iteration and the deriving of *SC* then SC^* may be derived via a sorting algorithm. Assuming a random distribution of *CR*s throughout *SC* (as expected with an *SU* of arbitrary ordering dictating the ordering of when objects become the *OUC* in the previous iteration) then an appropriate algorithm such as Quicksort that operates in $O(n \log_n)$ is required. However, in further iterations of the algorithm the coherence will become evident (as we assume objects move only small distances between each frame) and an expectation arises that the *SC* created is not a random distribution but closely resembles the SC^* from the previous iteration. Therefore, a more appropriate sorting algorithm would be Insertion sort as such sorting algorithms typically perform in O(n) for nearly sorted sets of data.

3.3 Ordered Axis

We now attempt to exploit coherence based on the sweep and prune approach. Usually, in this approach AABBs representing each object are checked for overlap. As we are dealing with spheres we have to alter the approach used for AABBs to suite our needs. However, the principle notion of sorting a nearly sorted list (axis) to exploit coherence is still the same.

A single axis is chosen to provide the basis of our algorithm. This choice is arbitrary but could be dictated by the nature of the simulation (e.g., the x or the y axis may be more suitable than the z axis for objects commonly located on or near flat terrain). For ease of explanation we chose the x axis to use in the examples presented here. The start and end points of an object, say O_a, are determined via $O_a^{pos} - O_a^{rad}$ and $O_a^{pos} + O_a^{rad}$ respectively. All x axis start and end points associated to all objects in *SU* are placed in the sequence OS^* and ordered in ascending order. It is now possible to identify which objects overlap in the x axis in the same manner as described in sweep and prune algorithms. Objects that share overlap in the x axis are determined to be members of an overlap relation (*OR*). We continue the example we have used throughout the paper and show in figure 4 the overlap area that is used to determine *OR*s. There is a relationship between the membership of *OR*s and *CR*s: an *OR* may be equivalent to a *CR* or a superset of a *CR*. From this observation we may assume that a *CR* may be derived by only considering a single *OR*. Therefore, we may apply expanding sphere, or the ordered *CR* expanding sphere, to an *OR* to derive the *CR*s. This may provide an alternative way to reduce the number of pairwise comparisons required from the previous two algorithms.

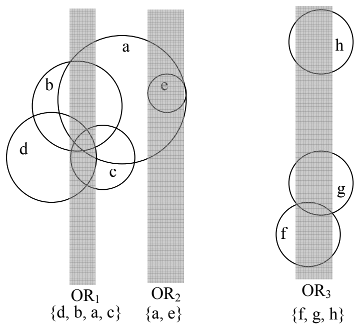


Figure 4 – Identifying overlap relations.

An ordered sequence of *OR*s (ORS^*) may be derived from OS^*. Each *OR* in ORS^* is inspected in turn and associated *CR*s are derived. Each inspection of an *OR* is simply a run of either one of the previous expanding sphere algorithms, with the *OR* assuming the role of *SU*. In principle, we are applying an expanding sphere algorithm on distinct sets (defined by ORS^*) of objects within the virtual world. Unfortunately, simply applying expanding sphere in this manner creates a problem. We alter our example slightly to explain this problem.

In figure 5 we have introduced three new objects (O_i, O_j and O_k). The introduction of these new objects has resulted in four *OR*s. Let us assume we have applied the expanding sphere algorithm to OR_1 and OR_2 and derived $CR_1 = \{O_d, O_b, O_a, O_c\}$ and $CR_2 = \{O_a, O_e, O_j, O_i\}$. We now apply expanding sphere to OR_3. Following the iteration of the expanding sphere algorithm taking OR_3 as the *SU* (input) we derive two *CR*s ($CR3 = \{O_i, O_j\}$, $CR4 = \{O_k\}$). Unfortunately, CR_3 is a subset of CR_2 and should not exist as a collision relation in its own right. Solving this problem is trivial and just requires that an object may not assume the identity of the *OUC* more than once during an iteration of the algorithm (across all *OR*s).

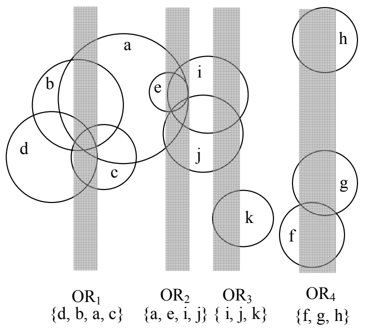


Figure 5 – Exploiting already determined *CR*s.

We may enhance this algorithm still further (reducing pairwise comparisons) by utilising *CR*s produced from earlier *OR*s. Applying expanding sphere to OR_3 with the remit that neither O_i nor O_j may become the *OUC* results in two comparisons (comparing O_k (the *OUC*) with O_j then O_i) resulting in $CR_3 = \{O_k\}$. This comparison may be reduced to one if we allow O_k to be compared with CR_2 (as O_i and O_j are both in CR_2). If we can dismiss CR_2 as not colliding with O_k then we may assume O_k does not collide with O_i or O_j. To accommodate this, a set of *CR*s may be maintained with which the *OUC* may be compared with. Such a set may only contain *CR*s that have members which also exist in the same *OR* as the *OUC*. In our example, if the *OUC* is O_k then this set would only contain CR_2 (as O_i and O_j are not present in any other *CR*).

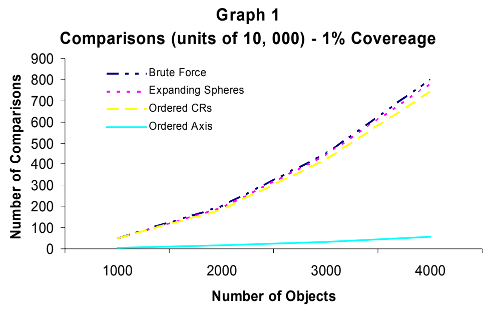
4. Performance Evaluation

Experiments were carried out to determine the performance of the expanding sphere algorithm, expanding sphere with ordered *CR*s and expanding sphere with ordered x axis. To enable comparative analysis of the performance figures, a brute force collision detection algorithm was implemented. However, it must be noted that the brute force algorithm does not provide the *CR*s as our algorithms do, this would require further processing above and beyond the brute force algorithm.

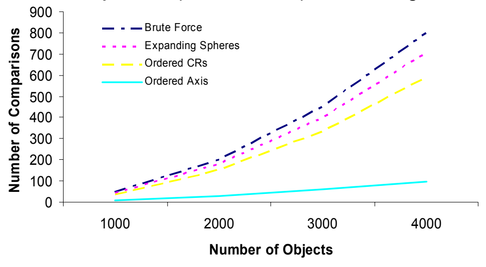
There are two performance measures which are of interest: (i) Number of comparisons; (ii) Number of frames achieved. The number of comparisons is a count of the number of pairwise intersection tests required to determine appropriate *CR*s for all objects between two consecutive frames of animation (brute force just provides pairings). This measurement may not be influenced by implementation details (e.g., hardware configuration, memory management). The number of frames achieved identifies the time taken for all *CR*s to be identified (a frame equates to one iteration of the collision detection algorithm). This measurement is influenced by the processing overhead of updating object positions. The time required for drawing a frame of animation is not a measurement we are interested in so is zero in our experiments (i.e., we draw no graphics).

We are interested in the scalability of our algorithms. Therefore, object numbers were increased gradually from 1000 to 4000 with the number of comparisons and time taken to derive *CR*s recorded at each increment. The experiments were repeated with 4 different levels of coverage via the resizing of the virtual world. Each level identifies the percentage of the virtual world contained within objects given that no two objects overlap (low density (1%), medium density (5%), high density (10%), and very high density (20%)). Experiments were conducted on a Pentium III 700MHz PC with 512MB RAM running Red Hat Linux 7.2 and all algorithms were implemented in C.

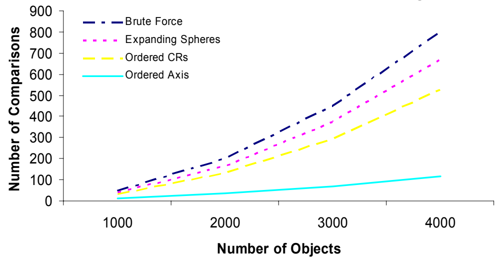
An attempt is made to provide realistic movement of objects within the virtual world. A number of targets (*T*) are positioned within the virtual world that objects (*O*) travel towards. Each target has the ability to relocate during the execution of an experiment. Relocation of targets is determined after the elapse of some random time (between T^t_{min} and T^t_{max}) from the time the previous relocation event occurred. Furthermore, objects may change their targets in the same manner (random time between O^t_{min} and O^t_{max}). Given that the number of targets is less than the number of objects and T^t_{min}, T^t_{max}, O^t_{min} and O^t_{max} are set appropriately, objects will cluster and disperse throughout the experiment. The objects are uniform in size and their size does not change throughout the experiments. Objects may pass through each other and may move freely in any direction.



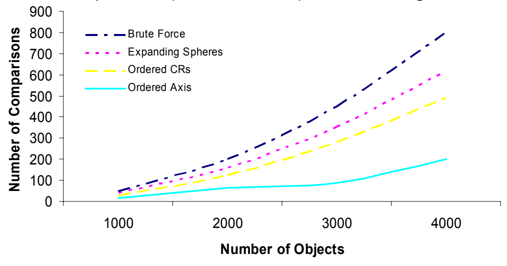
Graph 2
Comparisons (units of 10,000) - 5% Coverage



Graph 3
Comparisons (units of 10, 000) - 10% Coverage



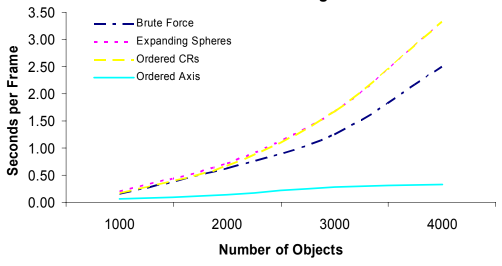
Graph 4
Comparisons (units of 10, 000) - 20% Coverage



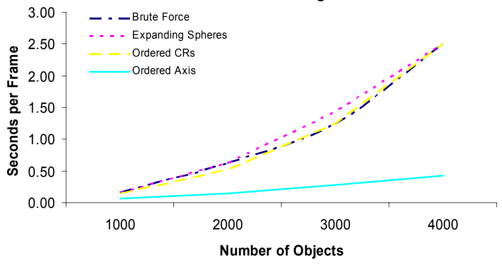
Graphs 1 through 4 identify the number of comparisons required to determine all *CR*s. The first observation to be made is that all variations of expanding sphere outperform brute force. This improved performance over brute force becomes more evident when the number of objects and coverage increases. From these observations, we may assume that our approach to determining *CR*s appear successful in reducing pairwise comparisons compared to brute force. When coverage is low (1%) we may assume that *CR*s are of a lower cardinality than when coverage is higher. Therefore, as coverage increases there is more opportunity to dismiss comparisons as *CR*s are more likely to be larger. This explanation appears appropriate for the similar performance expanding sphere and ordered *CR*s has compared to brute force.

The ordered axis variation of expanding sphere outperforms all the other approaches by lowering the number of comparisons significantly. Ordered axis requires approximately 12% of the comparisons associated to brute force to derive *CR*s. Furthermore, the rate of increase in comparisons of ordered axis is lower than all other approaches, appearing to scale nearly linearly.

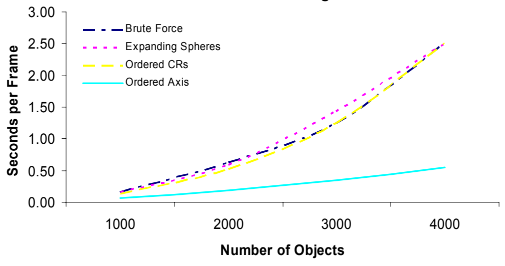
Graph 5
Frames - 1 % Coverage



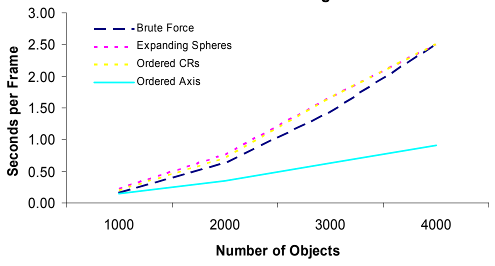
Graph 6
Frames - 5% Coverage



Graph 7
Frames - 10% Coverage



Graph 8
Frames - 20% Coverage



Graphs 5 through 8 identify how the number of seconds per frame scales with increasing numbers of objects. The expanding sphere and ordered CR variation of expanded sphere perform similar or slightly worse than brute force in all degrees of coverage. This indicates that expanding sphere is computationally expensive. Even though comparisons are lower in number (as identified in the graphs 1 through 4), the time taken to execute the algorithm is not much better than brute force. However, expanding sphere is producing sets of *CR*s which brute force is not and therefore this is not a fair comparison. A more promising observation is the performance of the ordered axis variation of expanding sphere. The ordered axis approach appears to scale nearly linearly. Even at 20% coverage, where there is a high expectation of collision and therefore more *CR*s, the ordered axis variation performs better than the other approaches.

5. Conclusions

We have presented an approach to collision detection that, we believe, is well suited to the aura based interest management schemes used in DVEs. Furthermore, we believe our algorithms to be suited to collision detection for non-solid moving spheres in more general cases of graphics processing.

Two variations of our approach that exploit coherence have been developed. We have shown that we can provide collision detection that scales nearly linearly when object numbers are increased beyond 1000. We are currently working on a distributed implementation of our algorithms for integration into our own DVE middleware services [16].

Acknowledgements

This work is funded by the UK EPSRC under grant GR/S04529/01: “Middleware Services for Scalable Networked Virtual Environments”. The authors would like to thank Dr. J. Steggles for his advice on algorithm descriptions.

References

[1] C. Greenhalgh, and S. Benford, “MASSIVE: A Virtual Reality System for Tele-conferencing”, ACM Transactions on Computer Human Interfaces (TOCHI), 2(3):239-261, September 1995

[2] D. Miller, J. A. Thorpe. “SIMNET: The advent of simulator networking”, In Proceedings of the IEEE 83(8), p 1114-1123, August 1995.

[3] T. Sweeney, “Unreal Networking Architecture”, http://unreal.epicgames.com/Network.htm, as viewed August 2003.

[4] C. Carlssom, O. Hagsand, “DIVE – A platform for multi-user VE”, Computer & Graphics 17(6), p 663-669, 1993

[5] M. H. Overmars, “Point Location in Fat Subdivisions”, Inform. Proc. Lett., 44:261-265, 1992

[6] P. M. Hubbard, “Collision Detection for Interactive Graphics Applications”, IEEE Transactions on Visualization and Computer Graphics, 1(3) 218-230. 1995

[7] S. Gottschalk, M, C. Lin, D. Monocha, “OBB-Tree: A Hierarchical Structure for Rapid Interference Detection”, SIGGRAPH 93, p247-254, USA, 1993

[8] M. C. Lin, “Efficient Collision Detection for Animation and Robotics”, In proceedings of the third Eurographics Workshop on Animation and Simulation, Cambridge, England, 1991

[9] J. D. Cohen, M. C. Lin, D. Manocha, M. K. Ponamgi, “I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments”, In Proceedings of the 1995 symposium on Interactive 3D graphics, pages 189–196, 218. ACM, Press, 1995

[10] D. Baraff, “Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation”, Computer Graphics, 24(4):19-28, August 1990

[11] D. J. Kim, L. J. Guibas, S. Y. Shin, “Fast Collision Detection among Multiple Moving Spheres”, IEEE Transactions on Visualization and Computer Graphics, 4(3):230--242, 1998.

[12] C. Greenhalgh, S. Benford, “MASSIVE: a distributed virtual reality system incorporating spatial trading”, Proceedings IEEE 15th International Conference on distributed computing systems (DCS 95), Vancouver, Canader, June 1995.

[13] C, Greenhalgh, S, Benford, “Boundaries, Awareness and Interaction in Collaborative Virtual Environments”, 6th Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises (WET-ICE '97), USA, 1997

[14] M. S. Paterson, F. F. Yao, “Efficient Binary Space Partitions for Hidden-Surface Removal and Solid Modeling”, Disc. Comput. Geom., 5, p 485-503, 1990

[15] S. Singhal, M. Zydra, “Networked Virtual Environments, Design and Implementation”, Addison Wesley, 1999

[16] G. Morgan, F. Lu, “Predictive Interest Management: An Approach to Managing Message Dissemination for Distributed Virtual Environments”, Richmedia2003, Switzerland, 2003