View Volume Culling
Using a Probabilistic Caching Scheme

Mel Slater and Yiorgos Chrysanthou
Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK.

Abstract
This paper describes a new algorithm for view volume culling. During an interactive walkthrough of a 3D scene, at any moment a large proportion of objects will be outside of the view volume. Frame-to-frame coherence implies that the sets of objects that are completely outside, completely inside, or intersecting the boundary of the view volume, will change slowly over time. This coherence is exploited to develop an algorithm that quickly identifies these three sets of objects, and partitions those completely outside into subsets which are probabilistically sampled according to their distance from the view volume. A statistical object representation scheme is used to classify objects into the various sets. The algorithm is implemented in the context of a Binary Space Partition tree, and preliminary investigation of the algorithm on two scenes with more than 11,000 polygons, suggests that it is approximately twice as fast as the hierarchical bounding box approach to culling, and that only about 14% of the total frame-polygons are passed through the viewing pipeline during the course of a walkthrough.

Keywords
Clipping, culling, graphics pipeline, BSP trees, virtual reality walkthrough.

1. Introduction

In the COVEN (Collaborative Virtual Environments) project (Normand and Tromp, 1996) we are developing a very large scenario for a virtual travel rehearsal within London. One of the problems that needs to be faced is that such large scale scenes cannot realistically be rendered at frame rates fast enough for real-time walkthrough and interaction - the data base is just too large. Several authors have addressed this issue in recent years (for example, Airey, Rohlf and Brooks, 1990; Teller and Sequin 1991; Teller, 1992; Naylor, 1992; Greene, Kass and Miller, 1993; Greene, 1996; Sudarsky and Gotsman, 1996). This paper concentrates on one particular aspect of the problem that we refer to as View Volume culling.

For any frame in an interactive walkthrough of a dynamically changing scene, the set of objects forming the scene data base is partitioned into three subsets: (I) those objects that are entirely inside the current view volume, (O) those that are completely outside, and (B) those that are intersected by at least one polygon belonging to the boundary of the view volume.

The aim of view volume (VV) culling is to efficiently identify these sets, O, I and B. Objects in set I may be rendered (without the necessity of clipping), those in B are rendered but require clipping, and those in C may be ignored. Clipping is an expensive operation, but in practice for a large scene, only a small fraction of the total number of objects actually intersect the boundary of the view volume at any frame therefore requiring clipping. Blinn (1991) remarked that clipping seems to be a wasted operation most of the time - having no visible effect on the rendered image; for the vast majority of objects, clipping will either determine that they needn't have been "clipped" at all (I), or remove them altogether (O). One of the advantages of VV culling is that it enables targeting of the clipping process toward only those objects that need to be clipped.

This partition of the scene data base changes dynamically for two reasons: the view frustum changes (translates, rotates), and objects move. However, in most circumstances there will be coherence from frame to frame based on spatial and temporal locality. Spatial locality implies that whichever of the three sets to which an object belongs in a particular frame, its neighbours are likely to belong to that same set. Temporal locality implies that whichever set an object is in during this frame, it is likely to be in the same set in the next frame. Since the sets correspond to regions of space relative to the VV, another way of expressing these localities is that objects tend to be in the same regions of space as their neighbours (obvious) and that from frame to frame they tend to stay in the same region of space. If we consider the region of space corresponding to the set O, we can also consider the partitioning of this space into subspaces which are classified according to their "distance" from the VV. The principles of temporal and spatial locality will apply also within these subspaces.

An ideal VV culling algorithm would determine the partition of the objects into these sets efficiently, based on temporal and spatial locality, and during the rendering process objects in O would not be touched at all. Avoiding references to objects in O could be important, since for a very large data base objects in O may be stored on disc, so that accessing them may be extremely expensive. Of course, however, it is not possible to avoid all references to O, since the set membership would change over time thus requiring an iteration through the data base in order to check for possible changes of allocation of objects to the sets.

One way to avoid an iteration through all objects is by exploiting hierarchy (Clark, 1976; Rohlf and Helman, 1994). For example, with hierarchical bounding volumes, as soon as any volume is determined as being wholly in I or O, then all of its children must be in the same set. The problems here are the overheads in building the hierarchy, and more importantly, of maintaining it as the scene itself changes through time.

An alternative approach is to subdivide world space into a uniform grid of voxels, with each voxel maintaining a list of

identifiers of objects that intersect it. Objects belonging to sets I and B can easily be determined, since at any time a set of voxels corresponds to the VV, and only objects within this set need to be rendered. In fact it is possible to differentiate objects belonging to I or B in this scheme. However, there are several drawbacks to this type of space partition approach:

- If the subdivision has low resolution (i.e., a relatively small number of voxels) then there may be large errors in the list of candidate objects for clipping.
- If the subdivision has high resolution then a large memory overhead is required, and also there is a higher cost of accurately computing the set of voxels corresponding to the view volume and the sets of voxels corresponding to objects.
- There is substantial cost in computing an accurate set of voxels corresponding to the view volume.
- Every voxel intersecting the view volume must be examined (there could be a large number of these) in order to determine if there are objects present. This can be expensive especially when objects are not uniformly distributed throughout the space. On the other hand an adaptive (e.g., oc-tree) method of subdivision can lead to a heavy cost in tracing the objects and view volume through an irregular space partition.

Nevertheless, this approach has the characteristics of a genuine VV culling algorithm in that it can in principle satisfy the fundamental requirement - to process only those objects inside or on the boundary of the view volume, and to not touch at all those that are outside the view volume. The problem with the space subdivision approach though is that it has undesirable characteristics from the point of view of implementation.

In practice VV culling would be combined with visibility culling. The latter is concerned with exploiting visibility relationships amongst objects in order to render only those fragments of objects that are actually visible in a given frame. This technique is most often used in architectural walkthrough, where there are natural subdivisions of the overall space (Airey, Rohlf, and Brooks, 1990; Teller and Sequin 1991; Funkhouser, Sequin and Teller, 1992). For example, in Teller and Sequin (1991) this subdivision is exploited by pre-computing a region-to-region visibility, and then during the execution of the walkthrough there is an eye-to-region visibility computation. Other methods have exploited partitioning trees, such as oc-trees and Binary Space Partition (BSP) trees. The use of an oc-tree in conjunction with a hierarchical Z-Buffer was introduced by (Greene, Kass and Miller, 1993; Greene, 1996). BSP trees were used by Naylor (1992) who incrementally constructs a BSP tree of the image using a traversal of the scene BSP tree from the viewpoint. Subtrees of the scene BSP tree that have not yet been traversed are compared with the image BSP tree, and if such a subtree can be shown to be already completely covered by a region represented in the image tree, then it is discarded.

Visibility culling is beneficial in the case of scenes that are heavily partitioned by objects such as walls that do obscure most of the scene "behind" them. Its impact is lessened for environments that have a large number of smaller objects (for example, as might be found within a complex room). However, (Coorg and Teller, 1996) have started to address how this problem might be tackled in the context of visibility culling.

In the VV culling algorithm presented here, at any moment objects are classified into the three sets I, B and O, and the set O is partitioned into subsets O_1, ..., O_n based on the distance of their contained objects from the view frustum. The objects in set O_i are sampled in each frame, with probability of selection inversely proportional to i, and checked to examine if they have changed set. The probabilities are updated according to changes in position and orientation of the view frustum, so that when the view frustum is moving away from the region corresponding to a particular set, its corresponding sampling probabilities are reduced. Moreover we employ a statistical representation scheme for objects which allows for efficient computation of the relationship between objects and the planes forming the view frustum.

In Section 2 we define the statistical representation of objects, and how this is used. The main algorithm is given in Section 3, followed by implementation details in Sections 4 and 5. Discussion and conclusions follow in Sections 6 and 7.

2. Statistical Object Representation

Objects are represented by their statistical properties, in particular their mean vector and covariance matrix. We adopt a paradigm that supposes that objects are in an ideal sense probability distributions over a vector-valued random variable (x,y,z) (which has, for example, a tri-variate Gaussian distribution). Samples from this particular distribution would be points in 3D space, and a large number of points would form an ellipsoid cluster with decreasing density towards the periphery. In this paradigm any region of space has an associated non-zero probability of containing a part of the object - so that theoretically, every object is distributed over the whole of space, though with greatest probability density corresponding to where the object actually "is" (in the normal sense). A 2D analogue is shown in Figures 1 and 2.

A probability distribution can be completely characterised by its moments, provided that these are all finite. In practice, the first two moments (the mean and variance for a scalar variable) are very informative, providing information about location, dispersion and correlation between x, y and z. Hence an object can be approximated by its mean vector (location) and covariance matrix (dispersion and correlation). This idea has been used in (Gottschalk, Lin, and Manocha, 1996) where the eigenvectors (or principle components) obtained from the covariance matrix are used to determine an object's oriented bounding box. In our algorithm the object is approximated by a bounding ellipsoid.

The following are some important relationships that we use in the construction of the algorithm. Let μ_p be the mean vector for the random variable p, and Σ_p the covariance matrix. The main diagonal of the covariance matrix consists of the variances of x, y and z respectively, and the off-diagonal terms the covariances. In the case of a Gaussian distribution, if all the covariances were zero and all the variances equal, then a large number of points sampled from this distribution would form a spherical shape.

Suppose that the object is transformed by an affine distribution, so that any point p becomes $q = pA + v$ where A is a 3×3 non-singular matrix and v is a translation vector. Then $\mu_q = \mu_p A + v$ and $\Sigma_q = A^T \Sigma_p A$, where A^T is the transpose of A.

Let $ax + by + cz + d = 0$ be a plane equation, with normal $n = (a,b,c)$. Then the perpendicular distance D of any point p to the plane is $(n^T p + d)/|n|$. If we consider p to be a random point on an object distribution, then the mean and variance of the distribution of the scalar *distance* D of that object from the plane is given by:

$$\mu_D = (n^T \mu_p + d)/|n|$$
$$\Sigma_D = n^T \Sigma n/|n|$$

A fundamental observation for our algorithm is that we can use this result to characterise the likely relationship of an object to the (typically) six planes of the view volume.

Suppose that $(n^T p + d) = 0$ is the oriented equation of a clipping boundary, where normal n points in the "outside" (or "invisible") direction. Then if for an object we find $\mu_D > 0$ (the mean distance of this object from the plane is positive) this indicates that the object is likely to be outside of that clipping plane, and similarly, inside when $\mu_D < 0$.

In particular, if the distribution is taken to be Gaussian as mentioned above, then the distribution of D is also Gaussian, and approximately 95% of the distribution of D lies within bounds $\mu_D \pm 2\sigma_D$ where $\sigma_D = \sqrt{\Sigma_D}$.

Up to now we have not specified object representation. However, suppose as usual that our actual representation of objects is as polyhedra. Then we estimate the mean vector μ_p by taking the means of the vertices of the polyhedra, and the covariance matrix by forming the estimated covariance matrix:

$$S_{ij} = \Sigma\Sigma(u_i - m_i)(u_j - m_j)/N \ (i,j = 1,2,3).$$

where N is the number of vertices, u_1 is the x-variate, u_2 is the y-variate, u_3 is the s-variate with m_1, m_2, m_3 being the corresponding estimated means.

Our paradigm inherently considers objects as solids, whereas the observations are only on the boundaries. Hence the estimation of the variances including Σ_D will be inflated. For the purposes of the algorithm we are interested in finding the clipping plane (if any) for which the (positive) mean distance is maximum. This corresponds to the clipping plane which the object is "most outside". (For example, the object could be inside all planes except for the far and right clipping planes. In such a case four of the mean plane distances would be negative, and two positive. We are interested in finding the maximum positive distance, and the corresponding plane). For this purpose, instead of using μ_D, which does not take account of the possible variation in D, we use $\mu_D - k.\sigma_D$ as the measure of distance (for some positive constant k - for example, in the ideal Gaussian case k would be 2 to capture 95% of the distribution). If $\mu_D > 0$ then obviously $\mu_D + k.\sigma_D > 0$, so we do not need to consider this side of the interval. On the other hand if $\mu_D - k.\sigma_D > 0$, then this is strong evidence that the entire object is outside of the clipping boundary, whereas if $\mu_D > 0$ but $\mu_D - k.\sigma_D < 0$ we would take this as evidence that the object at least partly intersects the boundary.

To summarise this section:

- We "idealise" objects as tri-variate continuous (point-mass) probability distributions in 3D World Coordinate space.
- An object is characterised by its mean vector and covariance matrix, which provides information about location, dispersion and orientation (i.e., the covariances amongst x, y and z).
- The relationship between an object and the view volume is characterised by the plane if any which has the maximum distance from the object, measured by $\mu_D - k.\sigma_D$ amongst those planes for which this value is positive.

We next use these results to present the algorithm.

3. Concepts of the Culling Algorithm

3.1 The Basic Algorithm

The algorithm maintains three data structures: (I) the set of identifiers of objects that are completely inside the view volume (and which therefore do not require clipping). (B) the set of identifiers of objects that intersect at least one boundary (and therefore require clipping).

The third data structure is a two dimensional array: for each of the six clipping boundaries we have an array of sets of identifiers of objects, where the members of a set have their "distances" (as defined above) within a certain interval. These intervals are constructed using a "width" (w) which would be chosen to be greater than the expected or typical distance of a camera move from frame to frame. Let B[i,j] be the set of objects corresponding to plane i (i = Left, Right, Bottom, Top, Front, Back) at level j. The levels are characterised by $(j-1)w \leq$ distance $< jw$ $(j = 1,2,...,L)$ where L is the number of levels. We also refer to the elements of this array as "cells".

Now any object will either have been determined to be in the visible set (I), or in the boundary set, or at least one of its distances from the clipping planes is positive. In this case find the plane (i) for which this distance (d) is a maximum (Figure 3), and put the identifier for the object into the sequence B[i,j] where $(j-1)w \leq d < jw$.

Finally, associated with each B[i,j] is a value $p_{ij} \in [0,1]$, representing the probability that an object in this set will be selected for checking to determine whether it has moved into another set (another plane, or another level). These probabilities are constructed so that within each plane i they are inversely proportional to j. A first overview of the algorithm is as follows:

Algorithm

Find the estimated mean vector and covariance matrix for all objects using the object vertices. This step is only carried out once, since if an object is transformed, the new mean and covariance matrix can be found by transformation.

Render all objects, recording which ones are actually clipped, forming the initial sets I and B.

For each camera change:

1. Obtain the new clipping planes in World Coordinates.
2. Update the probabilities p_{ij} according to how the VV has changed.
3. For each plane i, for each level j, use the p_{ij} to select objects for checking, and
move those objects to their new sets as required. Objects that appear to move into set I are never moved directly into I but into B first of all.
4. Render all objects in I without clipping, but checking if any should be moved to B.
5. Render all objects in B with clipping, but checking if any should be moved into I or into the B[i,j].

For any object that is transformed:

a. Transform its mean vector and covariance matrix.
b. If it is in B[i,j] check it and move it to the appropriate new set.

3.2 Updating the Probabilities

An important aspect of the algorithm is that the probabilities adapt to how the viewpoint (and therefore view volume) is changing - so that, for example, when the VV is moving away from a set of objects their selection probability is decreased, and when moving towards the set it is increased. Such updating must take account of rotation of the view volume. One way to do this would be to keep an overall mean and covariance matrix for all objects in, for example, B[i,_], and use these to compute a corresponding $\mu_D - \sigma_D$ with respect to plane i. Suppose d_k and d_{k+1} are these distances for two successive camera specifications, then the change $d_{k+1} - d_k$

could be used to update the probabilities p_{ij} (j=1..L). However, this scheme would require a lot of computation to update the overall mean vectors and covariance matrices whenever objects joined or left a particular cell.

Another possibility would be to base this on an arbitrary single object in a cell. The computation would be light and necessary anyway, but in the case of rotation, if this object happens to be near the part of the plane that is rotating towards it, then the distance would have seen to have decreased, but the opposite should be the case for objects that are have moved away from the plane because of the rotation.

Instead we use the following scheme. We define a virtual object that initially corresponds to the view volume expressed in World Coordinates. This object never changes. However, as the camera changes and the actual view volume changes, the changing distances of this virtual object from the planes of the view volume are used to update the probabilities. This takes into account not just change in actual distance of one view volume boundary from its successor, but also changes in orientation. The particular scheme used is described in the next section.

4. Implementation Heuristics

In this section we present some of the heuristics used for the current implementation. First we consider the probability scheme. Initially, each B[i,_] is assigned a probability of prob = ρ, and this is halved for each successive level. Let $\Delta d_k = d_{k+1} - d_k$. Then if prob is the current probability for a particular cell, we use prob := prob*(1 - Δd_k/w) as the updating formula (but clamped to the interval [0,1]). Hence when the distance increases the probability of selection decreases, and vice versa.

Next, the cells do not actually contain sets, but sequences, and we do not actually do a random selection within a cell, but start from the beginning of the sequence. For example, if in cell B[i,j] there are N_{ij} objects, and the probability is p_{ij} then we take the first $\lceil N_{ij} \times p_{ij} \rceil$ (ceiling) objects (so that always at least one object is checked). When a selected object is checked, it is moved to the tail of the sequence to which it has been assigned. Hence, if it stays in the same cell, it is simply moved to the back of that cell.

This has several potential advantages: First, there is not the computational overhead involved in "random selection from a set". Next, there is no special reason to suppose (initially) that the first object assigned to a cell is special in any sense in terms of its relationship to the clip region boundaries, compared to any other object in that cell - so the first might as well be selected. Third, there is also a possible "principle of locality" that is being exploited here - it is likely that in scene construction objects are not defined in arbitrary order, but that there is some organisation to the construction. Hence this scheme may exploit the fact that whatever happens to one object is likely to happen to its neighbour in any representation scheme.

Finally, if it is found that the last object checked in a sequence has changed, then we continue to check along the sequence beyond $\lceil N_{ij} \times p_{ij} \rceil$, until the first non-changed object is encountered, or the end of the sequence is reached.

5. A BSP Tree Implementation

The most natural implementation of this algorithm is in the context of a Z-buffer visibility algorithm. In fact this is the version of the algorithm discussed above, apart from one problem. For the algorithm to be most effective, clearly objects in set I should be rendered without any clipping, with clipping only used for objects in B. Systems such as OpenGL do not support such a possibility - for clipping to the view frustum is always enabled. Moreover, it is not possible in the context of such standard rendering systems to perform special purpose operations - such as not actually clipping an object but at least checking if any of its vertices are outside the clip boundaries. Of course this can be implemented, but the implementation cannot make use of the renderer, so that much of what the renderer does has to be duplicated.

Our initial implementation was, therefore, in a more challenging environment, that is, in the context of a Binary Space Partition (BSP) tree visibility algorithm (Fuchs, Kedem and Naylor, 1980; Fuchs, Abram and Grant, 1983). A BSP tree may be constructed by choosing a root polygon, and using its (oriented) plane to form a partition of all other polygons in the scene into three subsets: those on the "front" side of the root, those on the "back" side, and those on the same plane as the root. Polygons that intersect the root plane are split and assigned appropriately. The subdivision process is then applied recursively to each of the front and back sets. The BSP tree allows a back-to-front ordered display of the polygons in the scene: for any subtree, if the viewpoint is in the front half-space defined by the plane of the root then display the subtree referenced by the back child, then render the polygons at the root, and finally display the subtree referenced by the front child. If the viewpoint is in the back half-space, then first display the front subtree, then the root node, and then the back subtree.

The VV culling algorithm operates at the level of objects (i.e., polyhedra) rather than individual polygons. However, each polygon maintains a reference to the object to which it belongs. Therefore, when a polygon is encountered during traversal of the tree while producing a new frame, its corresponding object can be accessed. The object stores information about its visibility status - whether it is in the completely visible region, the boundary region, or in the invisible region (i.e., one of the cells). If the polygon is in the invisible region, it is not rendered. If it is in the visible region, it is rendered without clipping, but during this process there is a simple check to determine whether it should be clipped. If it is in the boundary region, it is rendered with clipping, and as a result there would be information about whether or not it was actually clipped by the boundary.

It is not so straightforward as described, however. Suppose that a polygon is in the boundary region, it is rendered and clipped, and the clipping determines that its status has changed (it is now completely inside, or completely outside). At this time the object status cannot be changed, for there may be subsequent polygons in the BSP tree belonging to this object. In order to determine whether or not the entire object has been processed while traversing the tree, each object also maintains an identification number (PIN), and there is a PIN also associated with each frame. Initially, all objects have the same PIN as the first frame. The frame PIN is incremented with each successive frame. When an object is encountered that has a different PIN from the frame PIN, this indicates that it is the first time that this object has been encountered for the current frame (in other words all of its polygons would have been processed during the previous frame).

Table 1
Updating the visible and invisible flags associated with an Object: Results of Clipping an Individual Polygon

Completely Outside	**Intersected by Boundary**	**Completely Inside**
visible = false	visible = false	visible && = true
invisible &&= true	invisible = false	invisible = false

Two boolean flags are stored for each object. The first is called "invisible", which has the value true if and only if all polygons so far processed for this object have been in the invisible region. The second is called "visible", which has the

value true, if and only if all polygons so far processed have been in the visible region. Both are initially set to true. The method for updating these flags is shown in Table 1.

Suppose all polygons of an object in the boundary region had been in the visible region - then visible would have the value true, invisible would have the value false, and the object may be reassigned to the completely visible region (I). If all polygons were in the invisible region, then visible would have the value false, and invisible would have the value true, and the object may be reassigned to one of the cells in the invisible region. Finally, if some polygons were visible, and some invisible, so that, of course, some would also be intersected by the boundary, then both flags would return false. In this case the polygon would be displayed with clipping, and the object would remain in the boundary region.

In fact, for the BSP tree implementation there is no need to keep an explicit set corresponding to the boundary region, since all objects are in any case visited (though obviously not necessarily rendered) during traversal of the tree. In one sense, since all polygons are "touched" during traversal of the tree, this does not represent the ideal for a VV culling algorithm - which should only consider at all objects in the visible and boundary set.

6. Results

The BSP tree version of the algorithm was tested in order to obtain information about two aspects of performance: first, the extent of the error which the algorithm produces, and second, its time performance compared with the hierarchical bounding box approach.

A test scene (S1) was a representation of a laboratory in the Computer Science Department at UCL. This consisted of 307 objects composed of 11,752 initial faces, and the size of the corresponding BSP tree was 23,648 faces. (The tree was constructed using Fuch's method as described in (Fuchs, Abram and Grant, 1983)). A random walkthrough was constructed by pre-selecting a sequence of points (uniformly) at random inside the overall bounding box of the scene, and the camera moved in small increments along the straight lines joining successive points, with its view plane normal vector corresponding to the direction of the line segment. At the end of such a line segment, the camera would interpolate through the angle between the previous line segment direction and the next, so that once again, its view plane normal vector would face along the current line segment. So the simulation consisted of the camera flying around the scene (translating and rotating) much as a virtual reality observer might fly around such a scene.

There were 588 such camera updates, and therefore frames, altogether, so that the total number of frame-polygons involved was $588 \times 23648 = 13{,}905{,}024$.

Table 2
Number of Polygons Processed

Polygons Processed	Frequency S1	S1 %	Frequency S2	S2 %
Displayed without clipping	627,258	34	1,440,015	49
Displayed with clipping	1,220,648	66	1,524,831	51
Total	**1,847,906**	**100**	2,964,846	**100**
Number Processed	1,847,906	13.3	2,964,846	15.3
Overall total	**13,905,024**	**100**	**19,394,550**	**100**

A second test scene (S2) consisted of a two layers of this laboratory, one on top of the other. This had 23,448 initial faces and 45.850 in the BSP tree. For the second animation the number of frames was 423, and the total number of frame-polygons was 19,394,550.

Table 2 shows the overall results. Out of the total number of frame-polygons, only 13% and 15% were actually passed down the viewing pipeline (i.e., rendered with or without clipping). Table 3 considers only those polygons that were clipped. Approximately one quarter of these turned out to be completely inside the view volume, whereas approximately 10 per cent intersected the boundary.

Table 3
Classification of Clipped Polygons

Clipped Polygons	Frequency S1	S1 %	Frequency S2	S2 %
Completely visible	285,382	23	440975	29
Completely invisible	830,771	68	935749	61
Intersecting boundary	104,495	9	148107	10
Total clipped	**1,220,648**	**100**	**1,524,831**	**100**

Table 4
Error Rates

Category of Error	Frequency S1	% total S1	Frequency S2	% total S2
Incorrectly not clipped	7,519	0.05	7,519	0.04
Visible but not displayed	32,542	0.23	32,542	0.17
Unnecessarily clipped	1,137,992	8.18	1,137,992	5.87

Table 4 shows error rates. The algorithm may clip polygons unnecessarily (because they were completely outside or completely inside the view volume). This was the largest category of error, but also the "safest" error to make (since the rendered scene is not incorrect if this occurs). The next largest category, is more serious, that is where a polygon is visible, but was not displayed at all. Finally, the case where a polygon should have been clipped but was not, was the category of error that occurred the least.

An attempt has been made to calibrate the algorithm with respect to the parameters:

- k - the multiplier used to measure distance μ_D - $k.\sigma_D$ (for these results k = 1.5);
- ρ - the initial selection probability assigned to each boundary (here 0.1 was used);
- w - the width of each cell (5 times the camera step distance);
- L - the number of levels for each boundary (L=10).

Trial and error indicated that a value of k of about 2 gives good results amongst a number of object shapes, where by inspection, the ellipsoid approximating the object fitted the object. The above experimental data used a value of k = 1.5, which provides too small an approximation for many shapes. However, this ellipsoid approximation is not accurate where there is an uneven distribution of vertices around the object - in this case the density is distributed closer to the region of the greatest density of vertices. This was pointed out by

(Gottschalk, Lin, and Manocha, 1996) who attempted to improve this situation for their oriented bounding box approximation by evening out the data distribution. In the context of collision detection this is important, but less important for VV culling - where anyway the scheme is probabilistic for rapid walkthrough. However, this is an area where, after the event, we have realised that an alternative scheme could be used. This would involve making a direct estimation of the best fitting ellipsoid that covers all or most of the vertices, and then using this ellipsoid to derive the mean and covariance matrix for the object. Unlike the case of real statistical estimation, we are not interested in the 'true' mean and covariance matrix of the object distribution (which anyway is only a fiction) but in using an idealised representation that has useful properties for rapid computation. We are investigating this alternative approach at the time of writing, which should further reduce the error rates (which are already very small).

We carried out a multiple regression analysis to examine the influence of the parameters ρ, w and L. The design had all 27 combinations of three levels of each of the parameters, ρ (= 1/9, 1/3, 1/2), w = (1, 2, 5) times the camera step distance, L = (5, 10, 20). There was no significant influence of any of these three parameters over the overall time performance of the algorithm. However, w and L (but not ρ) were highly statistically significant with respect to the error rate, in the case of both those incorrectly not clipped, and those incorrectly not displayed. In each case, over the range of data considered, the error rate was linearly and negatively correlated with both w and L, though L always the most significant. (In each case the square of the correlation coefficient, that is, the variation in the error rate explained by the linear model, was approximately 80%).

Table 5
Profile Times for the Hierarchical Bounding Box and Probabilistic Culling Algorithms

S1

	Bounding Box	**Probabilistic**
Overall Time	137s	89s
No. of times clip called	2,580,314	1,224,708

S2

	Bounding Box	**Probabilistic**
Overall Time	232s	139s
No. of times clip called	2,470,686	1,541,888

Table 5 shows the comparative timings, of two culling algorithms, the hierarchical bounding box, and the probabilistic culling algorithm. The time is the total time for the scene walkthrough, but excluding actual rendering time (the rendering occurred on an X11 server running as a separate process). The profiling was carried out on a SUN Ultra 2 UPA/SBus (2 X UltraSPARC 168mhz) with 192M available memory, under SunOS Release 5.5.1. The table also shows the number of calls to the clip function generated by the two algorithms. The new algorithm, for these particular scenes and these implementations completes the animation in about 60-65% of the time of the bounding box method, and with about 50-60% of the calls to the main clip function. However, it is to be emphasised that these are results, for one type of scene only, and one set of parameters controlling the operation of the new algorithm.

7. Conclusions

This paper has introduced a new approach to view volume culling - a method that attempts to reduce the amount of unnecessary computation involved in clipping objects - which are anyway completely visible or invisible. It achieves this by maintaining a series of object caches, sets of objects that are likely to be completely outside the view volume, and which are probabilistically sampled at each new frame. The probabilities are inversely proportional to the distance of the regions corresponding to the caches from the current view volume.

A statistical representation is used for objects, which although approximate, captures the locality, spread and orientation of an object. This statistical representation is suitable also for rapid updating when objects are transformed, and may be used as a basis for computing a "distance" measure of an object from a clipping boundary plane. Moreover, an object can be represented by just nine floating point numbers (the mean and covariance matrix). This statistical method is, moreover, somewhat independent of the form of actual representation of objects - for example, it could equally well be used on B-Spline control points as on vertices of polyhedra.

The overall philosophy of the algorithm borrows from memory hierarchy in computer architecture - it employs temporal locality, in that objects are partitioned into sets which tend to remain constant from frame to frame. It achieves spatial locality with respect to polygons, since the scheme is object based - so that whatever computation is appropriate for one polygon, is likely to be appropriate for its neighbours (in the same polyhedra). It may also achieve an additional degree of spatial locality through the particular sampling scheme employed (where when an object is moved from one cell to another, its neighbour in the sequence is also checked for a possible move).

The algorithm has been tested for particular scene walkthroughs, and the results are encouraging. For these scenes it is significantly faster than the hierarchical bounding box method. Further work is continuing on estimation of a good mean and covariance matrix representation of objects, using an ellipsoid fitting method from which these parameters can be derived.

Acknowledgements

This work is supported under the COVEN Project, the European Union ACTS Programme, Project Number AC040. It has also benefited by support from Sun Microsystems.

References

Airey, J., Rohlf, J., Brooks, F. (1990) Towards image realism with Interactive Update Rates in Complex Virtual Building Environments, SIGGRAPH 1990 Symposium on Interactive 3D Graphics, 24(2), 41-50.

Blinn, J.F. (1991) A Trip Down the Graphics Pipeline- Line Clipping IEEE CG&A Jan, 11(1) 98-105.

Clark, J.H. (1976) Hierarchical Geometric Models for visible surface algorithms, Communications of the ACM, 19(10), 547-554.

Coorg, S., Teller, S. (1996) A Spatially and Temporarily Coherent Object Space Visibility Algorithm, Tech. Report MIT February 1996.

Funkhouser, T., Sequin, C. and Teller, S. (1992) Management of Large Amounts of Data in Interactive Building

Walkthroughs, in Proc. 1992 ACM Symposium on Interactive 3D Graphics, 11-20.

Fuchs, H., Kedem, Z.M., Naylor, B.F. (1980) On visible surface generation by a priori tree structures, Computer Graphics (SIGGRAPH), 14(3), 124-133.

Fuchs, H. Abram, G.D., Grant, E.D. (1983) Near real-time shaded display of rigid objects, Computer Graphics (SIGGRAPH) 17(3), 65-72.

Gottschalk, S., Lin, M.C., Manocha, D. (1996) OBBTree: A Hierarchical Structure for Rapid Interference Detection, Computer Graphics (SIGGRAPH Proceedings), 171-180.

Greene, N., Kass, M., Miller G. (1993) Hierarchical Z-Buffer Visibility, Computer Graphics (SIGGRAPH Proceedings), 231-238.

Greene, N. (1996) Hierarchical Polygon Tiling With Coverage Masks, Computer Graphics (SIGGRAPH Proceedings), 65-74.

Naylor, B. (1992) Partitioning Image Tree Representation and generation from 3D Geometric Models, Proceedings of Graphics Interface '92, (Vancouver), 201-212.

Normand, V. and Tromp, J. (1996) Collaborative Virtual Environments: The COVEN Project, Proceedings of the Framework for Immersive Virtual Environments Conference, FIVE'96, December, ed. by M. Bergamasco, Pisa.

Rohlf, J. and Helman, J. (1994) IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics, Computer Graphics (SIGGRAPH Proceedings), 381-394.

Sudarsky, O. and Gotsman C. (1996) Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality, Eurographics 96, J. Rossignac and F. Sillion (eds) Blackwell Publishers, 249-258.

Teller, S. and Sequin, C. (1991) Visibility Preprocessing for Interactive Walkthroughs, Computer Graphics (SIGGRAPH) 25(4), 61-69.

Teller, S. (1992) Visibility Computations in Densely Occluded Polyhedral Environments, PhD Thesis, University of California at Berkeley, Report No. UCB/CSD 92/708.

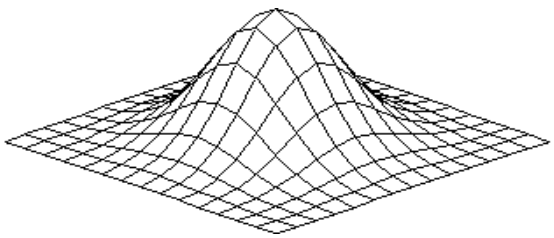

Figure 1 - Bivariate Gaussian Distribution

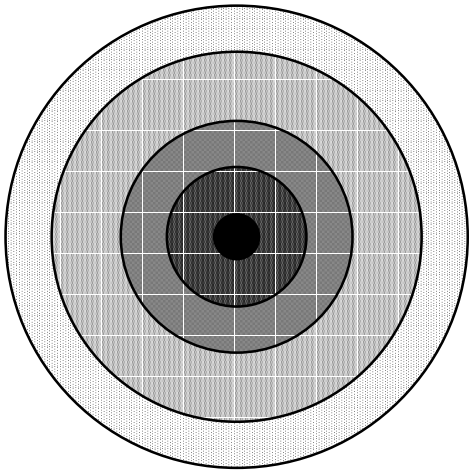

Figure 2 - Samples on Bivariate Gaussian
Observation Density Decreases Towards the Periphery

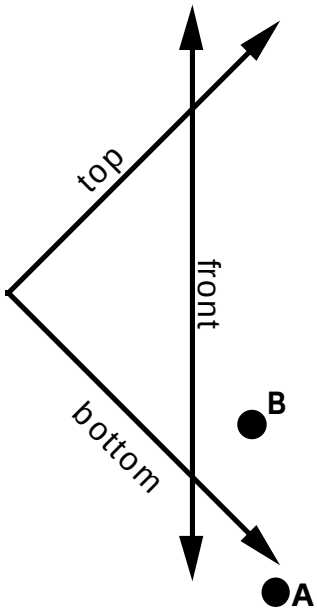

Figure 3 - Points Outside the Clip Boundary
Point A is assigned to the bottom plane, and B to front.