Directed Safe Zones and the Dual Extent Algorithms for Efficient Grid Traversal during Ray Tracing

Sudhanshu K Semwal Hakan Kvarnstrom
Department of Computer Science
University of Colorado, Colorado Springs
semwal@redcloud.uccs.edu

Abstract

Ray tracing is inherently a very time consuming process. There have been a variety of techniques developed for reducing the rendering time. While using space subdivision techniques, the object space is divided into a set of disjoint voxels; and only those objects are checked for intersection with the ray which pierce the voxels along the path of the ray. As the grid size is increased in an attempt to reduce the number of objects encountered along the path of the ray, more empty voxels are encountered along the path. In other words, considerable rendering time could be invested in moving from one empty voxel to another empty voxel. The proximity clouds method uses distance transformations to identify the empty regions, and combined with the 3DDA grid traversal (SEADS) technique, has been shown to be the fastest grid traversal technique available today.

In this paper, we present two further improvements to the proximity clouds method — called the Directed Safe Zones (DSZ) and the Dual Extents (DEs). We have implemented four methods for our comparison: SEADS (3DDA), proximity clouds, Directed Safe Zones (DSZs), and Dual Extents (DEs). We present both the theoretical and statistical analysis of the four methods. We show that DSZs and DEs surpass the performance of both the proximity clouds and SEADS implementations. The DSZs method *usually* outperforms the Dual Extents implementation, except when the topology of the scene favors the Dual Extent method.

Keywords: Rendering, ray tracing, grid traversal, space subdivision.

1 Introduction

Ray tracing is one of the most popular methods to create photorealistic images. Rays start from the eye or camera position and traverse through the scene to find the nearest intersection point. As the ray traverses through the scene it might intersect with objects, causing it to reflect or refract. Every time an intersection occurs the ray is recursively split into several new rays representing the reflection and refraction of the ray. Every pixel of the final image is rendered separately by casting a ray originating at the eye and passing through that pixel [1]. There are many techniques for reducing the image generation time [2, 3, 4, 5, 6, 11, 16, 17, 18, 19]. The space subdivison techniques are particularly useful as the nearest intersection point can be found without the need for testing intersection with all the objects. In this paper, our aim is to speed up the ray tracing process by quickly bypassing blank areas while using the grid method of space subdivison.

In space subdivision techniques, the space occupied by the scene is subdivided into small regions or *voxels*. Rather than checking a ray against all objects, we determine whether the region through which the ray is currently traversing is occupied by any object. If the region (voxel) is empty, the ray can pass through the region without looking for intersections. As shown in Figure 1(a), objects A and B are tested against intersection while object C is not. Several methods have been developed based on this principle. Some of them are: uniform spatial subdivision (SEADS, ARTS) [12] and the octree method [10].

The uniform spatial subdivision (SEADS) [12]

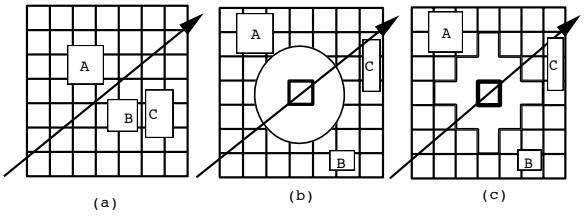


Figure 1: a) Grid method, b) Proximity (circular) cloud, c) city-block Proximity Cloud

method involves preprocessing of the scene to create a grid data structure (SEADS – spatially enumerated auxiliary data structure). Due to the uniform subdivision, the use of SEADS enables very fast ray traversal from one voxel to another. The ray traversal is realized by using the 3DDDA (3 Dimensional Digital Differential Analyzer) [12]. In this paper, our implementation for grid traversal is similar to Cleary and Wyvill [8] method and is not explained here due to space limitations.

When an oct tree spatial subdivision method is used for space-partitioning, each region (also called a voxel) can vary in size, depending on the topology of the scene. In this method, large regions of empty space or large regions that contain a single object are not subdivided to the same extent as areas containing several small scattered objects. The main advantage of octree is that the memory requirements are usually lower compared to a uniform subdivision scheme. Although the empty regions are bypassed quickly, the main disadvantage is that the octrees can lead to extremely unbalanced trees which can incur high ray traversal cost when the scene is populated with objects. The 3DDA ray-traversal is usually faster in comparison to the oct tree ray traversal, as integer math used during the ray-traversal more than compensates for the larger voxel size in the oct tree method. Fujimoto and Iwata [12] developed the ARTS method which uses a combination of grid traversal (SEADS) and an octree to speed up the ray tracing. The ARTS method was empirically demonstrated to be better than the octree method [12, 14].

To further reduce the number of objects encountered along the path of the ray, the size of the voxels could be decreased by increasing the grid size. However, rendering time for the grid (SEADS) implementation increases when grid size is increased (e.g. see Tables 3, 5, and 7). This is because, there are more (smaller) empty voxels which the ray must traverse. To effectively bypass these empty regions during ray traversal, the oct tree preprocessing technique is used in both the ARTS method [12] and the Modified Slicing Extent Technique (MSET) [14]. However, oct trees are inherently too constrained because of their top-down, pyramid [13] approach of isolating blank-regions. A better technique for isolating blank regions is the flat pyramid approach of [13] which uses distance transformation. Instead of going to the next voxel, the ray is moved a safe (larger) distance to bypass the empty region quickly. In the section below, we explain the idea of distance transformation. More details are in [13].

2 Distance transformations

Consider a two-dimensional grid of black and white (unoccupied) pixels. How do we calculate the distance from every white (unoccupied) pixel to the closest black (occupied) pixel? A very simple but computationally expensive approach is to start at every white (unoccupied) pixel and recursively scan all directions until we find a black pixel. A faster approach is described by Borgefors [9] based on the distance transforms. A distance transformation converts a binary image of featured (black) and non-featured (white) pixels into a distance map where all non-featured pixels hold a distance value to the nearest featured pixel. In the field of ray tracing, we can use distance maps to represent distances in between voxels. Voxels containing objects are considered featured and empty voxels are non-featured.

Different metric systems can be used when calculating the distance maps. The two most commonly used metrics are *the Euclidean-metric* ($D_e = \sqrt{(\Delta x)^2 + (\Delta y)^2}$), and *the city-block metric* ($D_{cb} = \Delta x + \Delta y$). The advantage of using city-block metric in favor of the Euclidean metric, is that the square-root operation is computational expensive and can be avoided if city-block metric is used. The disadvantage is that it is less accurate but it is still a decent approximation of the Euclidean distance. Figure 1 (b and c) show the benefit of using Euclidean and city-block distance approximation during ray tracing.

Generating a city-block distance map: The global distances of the cell grid are approximated by propagating local distances over the grid. The local distances are the interdistances between adjacent cells. Using the city-block metric, the interdistances are always 1 since the cells have unit size and are mapped onto an integer grid. Before we start to

propagate, the value of all the grid-cells are preset such that empty cells are initialized to infinity and cells containing objects to zero.

∞	1	∞
1	0	1
∞	1	∞

Figure 2: City-block distance mask.

The distance mask is propagated twice over the grid, once from left to right and bottom to top (Figure 4(a)) and a second time from top to bottom and right to left (Figure 4(b)). During the propagation, the global distance of the empty cells are updated by taking the minimum value of the current cells global distance and the propagated distance value, as follows:

$$v_{i,j} = minimum(v_{i,j}, M[k,l] + v_{i+k,j+l})$$

Where $v_{i,j}$ is the current distance value of the cell at position (i,j) and $M[k,l]$ is the mask value at position (k,l). This mask is split into a forward mask and a backward mask (Figures 2 and 3). During both forward or backward calculations, the mask's entry with zero value corresponds to the (i,j)th entry.

	j-l	j+0
i-1	∞	1
i+0	1	0

	j+0	j+1
i+0	0	1
i+1	1	∞

Figure 3: Forward (k,l = 0,-1) and Backward (k,l=0,1) distance Mask.

Initially, occupied (black) voxels are assigned a value of zero, and remaining un-occupied (white) voxels are assigned a value of ∞ (or a suitably large value) (See Figure 4). For the first propagation, distances are calculated using the forward mask. This propagates global distances from voxels already visited. When we propagate the second time, we use the backward mask to find the distance values. Basically, we are propagating the global distance by adding the local distance mask to the values of voxels previously visited. Figure 4(a) and (b) show the values after the first and the second propagation.

3 The Proximity Clouds Method

The proximity clouds method [13] uses distance maps to speed up the grid-traversal of the cell space.

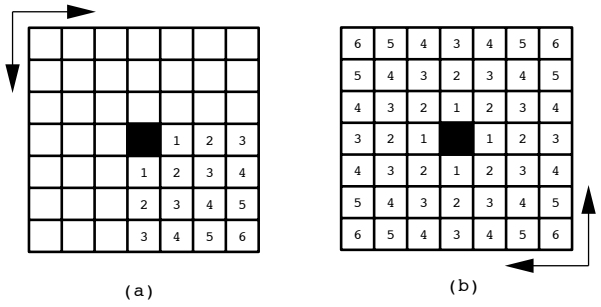


Figure 4: Distance map after first (a) and second (b) transformation.

It uses the fact that most of the cells of the grid contain no objects, e.g. when the level of subdivision is reasonably high and the scene has some degree of spatial coherence. The empty cells are used to store a distance value to the nearest non-empty cell. This distance defines a *free-zone* wherein no other objects reside and is also called a proximity cloud. Due to this safe-zone surrounding the cell, it is safe to jump or skip over the empty cells along the rays direction without missing a possible intersection with other objects. Cohen and Sheffer [13] show that city-block metric is the most effective, both in terms of speed and skipping distance.

Ray traversal using the Proximity Clouds: It is possible to safely skip over the empty voxels along the direction of the ray without missing a possible intersection. Please note that we have implemented all the methods in three-dimensions, however Figures in the paper show 2D-examples for simplicity and clarity. Assume we have a ray $R + tR_d$ connecting two points $R_1(x_1,y_1)$ and R_2 (x_2,y_2). Using the city-block metric, and algebraic manipulations in [15], we have

$$\begin{aligned} x_2 &= x_1 + D_{cb} C_x \\ y_2 &= y_1 + D_{cb} C_y \end{aligned}$$

Here, $C_x = \frac{R_{d_x}}{R_{d_x}+R_{d_y}}$, $C_y = \frac{R_{d_y}}{R_{d_x}+R_{d_y}}$, and $D_{cb} = \Delta x + \Delta y$. R_{d_x} and R_{d_y} are the x and y components of the direction, R_d, of the ray.

Note that it is only possible to proceed with skips as long we are not in the vicinity of objects. Once the skip distance falls below a predetermined value, we switch to the standard face-adjacent traversal in our implementation. This serves two purposes. First, if the distance is small, it is more efficient to perform grid-traversals due to the overhead of the skipping

algorithm. In our implementation, when D_{cb} is less than five, grid traversal is performed (this has proven to be a good solution for our implementation). Second we must avoid skipping beyond the free zone and therefore the skip value is D_{cb}-1, forcing a traversal to be performed just before entering a non-empty voxel.

Let n be the total number of skips performed. Then the sum of the total skip distance in the x and y direction respectively is

$$\Delta x = \sum_{i=1}^{n} D_{cb_x}[i], \quad and$$

$$\Delta y = \sum_{i=1}^{n} D_{cb_y}[i]$$

Cohen and Sheffer [13] tested this method on a discrete[1] ray tracer, yielding a more than 30% speed-up of the total execution time. Increasing the number of voxels will make the proximity clouds algorithm even more efficient compared to just using the grid (incremental step) traversal, because larger sized proximity clouds are generated.

In the following Sections, we present two new improvements to the basic voxel traversal algorithm.

4 The Directed Safe Zone (DSZ) Method

The proximity cloud method does not take the direction of the ray into consideration. The skip distance is always limited by the proximity distance even if the ray is facing away from the cell containing objects. A better approach would be to take advantage of the direction of the ray, and skip a distance defined by objects located in the direction of the ray. In R^2 this will create four zones, one for each edge of the cell (Figure 5).

These *safe zones* have a greater average distance than the proximity clouds. This is true because the proximity cloud distance can be extracted from the safe zones distances by taking the minimum of all distance values. In R^3, the algorithm uses six different distance values, one for each face of the voxel. During traversal we need to keep track of which face the ray will pierce when it leaves the current voxel. Notice that the skip algorithm is the same as for the proximity clouds method once we have determined which distance value to use.

[1] A discrete ray tracer uses a high resolution voxel grid where every voxel contains a single, precalculated intersection point and thereby eliminates the need for intersection tests during traversal.

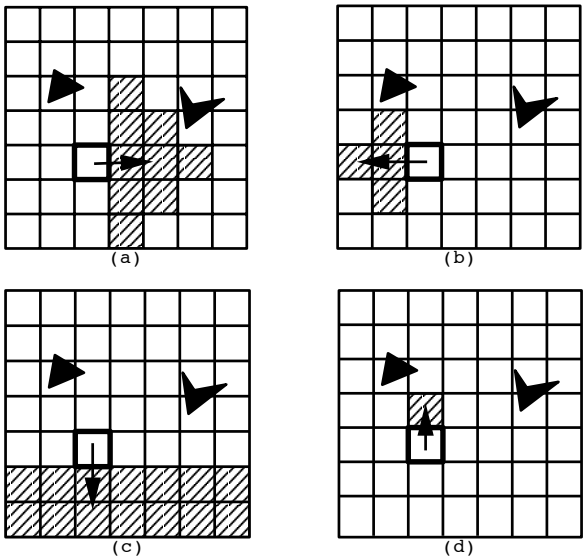


Figure 5: Directed Safe Zones: a) +x, b) -x, c) -y, d) +y directions

4.1 Directional distance transforms

The algorithm presented in Section 4.1 for creating distance transform is a linear time algorithm. Two passes through every voxel meant that the algorithm was O(N) where N is the total number of voxels in the grid. In this section we show an efficient algorithm to generate *directional* distance transforms using the city-block metric.

The process of creating a directional distance map of a cell grid is slightly more complicated than the non-directional distance maps. In the non-directional case, we had to scan all cells twice, propagating the minimum sum of all local and global distances. For the directional case, it is necessary to extend the scan directions so that it propagates global distance minimums towards all directions of interest. In R^2 we have to scan the cell grid from four different directions (Figure 6(a-d)): a) **Phase 1:** Top-to-bottom and left-to-right (D^1). Calculates negative x-distances and positive y-distances (Figure 6(a)). b) **Phase 2:** Top-to-bottom and right-to-left (D^2). Calculates positive x-distances and positive y-distances (Figure 6(b)). c) **Phase 3:** Bottom-to-top and left-to-right (D^3). Calculates negative x-distances and negative y-distances (Figure 6(c)). d) **Phase 4:** Bottom-to-top and right-to-left (D^4). Calculates positive x-distances and negative y-distances

(Figure 6(d)).

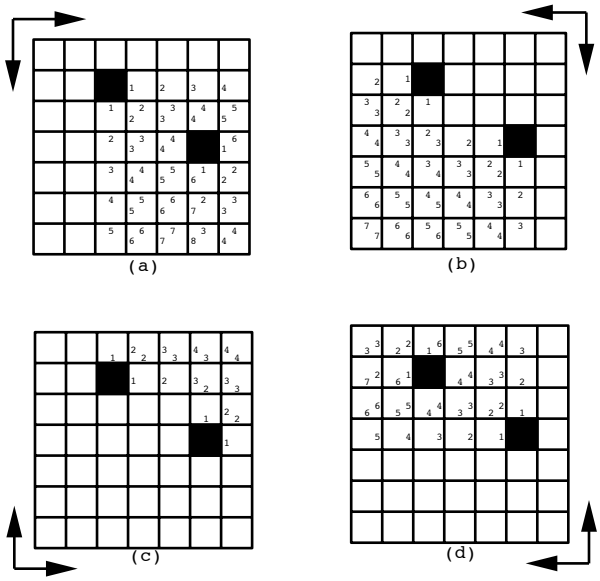


Figure 6: Calculation of the directed safe zone, a) phase 1, b) phase 2, c) phase 3, and d) phase4.

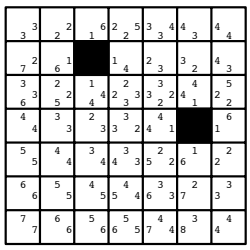


Figure 7: Resulting distance map of the directed safe zone.

Once we have the set of distances, we apply a minimum function to each of the cells direction values obtained by the four scan operations (Figure 7).

$$D_{xneg} = min(D^1_{xneg}, D^3_{xneg}) \quad (1)$$
$$D_{xpos} = min(D^2_{xpos}, D^4_{xpos}) \quad (2)$$
$$D_{yneg} = min(D^3_{yneg}, D^4_{yneg}) \quad (3)$$
$$D_{ypos} = min(D^1_{ypos}, D^2_{ypos})$$

Fortunately, we can perform the *min* operation at the same time as we propagate the cells. This eliminates the need to allocate extra memory to hold the distance map of each scan.

The extension to three dimension is trivial. Since we now have six different directions, we must extend the four steps above to include 8 steps. Four in the positive z-direction and four in the negative z-direction. In other words, we start a scan at each of the voxels 8 corners and propagate the global and local distances as before. The complexity of the algorithm is not dependent on the number of objects in the scene nor the number of voxels containing objects. It remains linear in terms of the number of voxel in the grid i.e. O(N) as each voxel is visited at most 8 times. As expected, directed safe zones use more memory than the Proximity Clouds method as six distance values are stored in each of the empty voxels compared to a single value using proximity clouds. As discussed in [15], an eight bit representation of the distance values works well. So we need only six bytes for storing the entire voxel's distance values. A comparison of the actual memory used during testing is in Tables 1 and 2.

Skipping algorithm: The algorithm to skip over the safe zones is identical to the one used when skipping over the proximity clouds (Section 3.1). The problem is to determine which one of the six distance values to choose. We know that the distance value is determined by the next face-adjacent cell the ray visits along its path and therefore we simply choose a distance value determined by the result of the traversal algorithm. Pseudocodes and other details of the traversal algorithms are in [15].

5 The Dual Extent Method

The dual extent method tries to find *lanes* of empty cells in the scene. Starting with a proximity cloud, we extend each of the six faces of the cloud until we hit a non-empty cell. This will allow the ray to skip a longer distance, especially if the ray is parallel or close to parallel to one of the coordinate axes. Figure 8(a) shows the dual extent for only the upper face of the proximity cloud. In addition to the skip resulting from the proximity cloud, we can now skip an additional distance defined by the boundary of the lane. Using a city-block metric, the length of the lanes is defined starting at the edge of the proximity cloud and ending at the closest non-empty cell within the lane boundary (See Figure 8(b)).

Generating the dual extent distance map is a little more involved compared to the proximity clouds. First we start by generating a proximity clouds distance map to be used as a starting point for the extents. Now, for each cell of the grid, we find the extents for each face of the proximity cloud. To help us find the solution to the problem, we represent the

proximity cloud as a height field. In Figure 8(b), we show the proximity cloud as a height field originating at the center of the proximity cloud. The height of each separate bar is defined by: $H[i] = D_{cb} - i$, where i is the integer distance to the center of the cloud and D_{cb} is the proximity cloud size. Note that negative values of $H[i]$ indicate that we are outside the proximity cloud.

The next step is to find the maximum extent of each bar. That is, how far, in the extent of each bar, can we skip without missing a non-empty cell? To find that distance, we scan the entire cell grid and propagate the distance values column by column to the closest object. This process is shown in Figure 8(c).

We start by creating a row $R_{ext}[j]$ to hold the propagated distance values. In the next step we initialize each cell of R_{ext} to infinity, or a value defined to represent infinity. Then for every corresponding non-empty cell of the row, we set the distance to zero. Propagate the distances to the next row by adding one to each cell. Once we have the distance value of each column to the nearest object, we can easily combine this with the height fields of each proximity cloud to get the final extent. The dual extent distance in R^2 can be expressed as:

$$D_{de} \quad = \quad R_{ext}[p] - H[i] - 1$$

Where p is the cell index of R_{ext} corresponding to i. D_{de} is calculated for every i within the boundary of the cloud. The final dual extent distance is the minimum of these values.

The extension to three-dimension requires R_{ext} to be calculated, not using a row, but a slice of cells. Each cell of the slice is then combined with the corresponding height value of the proximity cloud. Note that these height fields are a two-dimensional array of values. We now have:

$$D_{de} \quad = \quad R_{ext}[p, q] - H[i, j] - 1$$

The pseudocode for generating the dual extents and other details of the algorithm are not presented here, and can be found in [15].

Similar to the directed safe zone method, the dual extents uses a union construct to share the memory used by the distance values and the object pointers. Seven distance values need to be stored in every empty voxel. These distances, the proximity distance plus the six extents occupy a total of seven bytes.

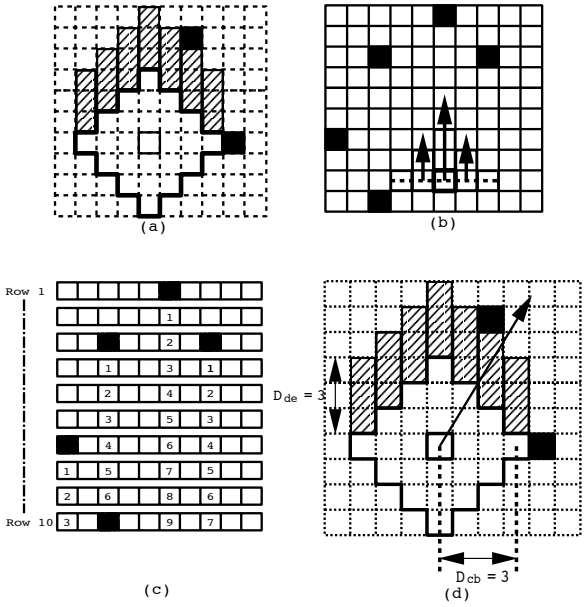


Figure 8: Dual extent: a) height fields, b) and c) row of distance values to the closest non-empty cell of the same column, d) proximity cloud and column generated.

Skipping algorithm: What is the greatest distance we can skip in the direction of the ray when using the dual extent method? If we simply add the proximity distance and the extent distance together, it is likely that we end up beyond the extent of the lane. We know that the sum of the two distances is the maximum distance a ray can skip if it is parallel to the lane. As soon as the ray is not parallel to the lane, the skip distance must be recalculated to fit the bounds of the lane (Figure 8(d)). Recall from Section 4.1 that the C_x and C_y split a distance S in $\Re^2$ into the distances along each axis. The split distances, let us call them S_x and S_y, are defined by:

$$\begin{aligned} S &= D_{cb} + D_{de} \\ S_x &= C_x * S \\ S_y &= C_y * S \end{aligned}$$

where D_{cb} and D_{de} are the proximity and dual extent distances respectively. Now if S_x or S_y is greater than D_{cb} (depending on the direction of the extent), we know that we have skipped too far and thereby we must be outside the bounds of the lane. This is true because the lane is defined as an extension of the proximity cloud and any perpendicular distance greater than D_{cb}, must leave us outside the

lane. Since D_{cb} usually is very small compared to D_{de}, most of the time we would have to recalculate S in our implementation so that that we do not skip beyond the wall of a lane. The overhead of testing whether the ray skipped too far or not is therefore wasted since, in most cases, we are forced to recalculate S. A better solution is to always recalculate S, no matter how small it is. The total performance would most certainly increase since most of the times it would have to be recalculated anyway.
Assume we have a proximity cloud of size D_{cb}, extended by a dual extent in the positive y-direction of size D_{de} (Figure 8d). The new skip distance S' is:

$$S' = \frac{D_{cb} - 1}{|C_x|}$$

In $\Re^3$ the new distance is:

$$S' = \frac{D_{cb} - 1}{(|C_x| + |C_z|)}$$

S' must never be greater than $D_{cb} + D_{de}$ thus

$$S'' = MIN(D_{cb} - 1 + D_{de}, \frac{D_{cb}}{(|C_x| + |C_z|)})$$

Notice that we have to calculate S'' using the proximity distance subtracted by one. This might seem a little strange but it is caused by the fact that we are using integer values to represent the cell values. As the ray traverses the scene it always holds an accumulated error-term. Since the distance maps are calculated using the center of the cells the error-term may cause the ray to miss a corner of a cell. The skip S'' can now be performed.

6 Theoretical analysis of logarithmic city-block distance metrics

Using a city-block metric D_{cb}, we can create distance transforms mapped onto an integer grid. Assume that instead of using the set of all positive integers Z^+ we use a subset of these integers which have the form 2^n. These distances, denoted by D_{lg2} can be expressed as: $D_{lg2} = 2^{\lfloor log_2(D_{cb}) \rfloor}$

These logarithmic distances will make the skip distance shorter than before since $D_{lg2} \leq D_{cb}$. The total number of skips that has to be performed are therefore higher. The main advantage is that a skip operation can be performed by using fast bit shifts instead of expensive multiplications. In addition, memory savings could also be realized. A detailed discussion on the logarithmic distances is in [15].

7 Implementation Platform and Scenes used for testing

Because of the space limitations, we would only describe the main points of our implementation here, for details please refer to [15]. We used the DKB-Trace package written in C by David Buck for implementing all the four methods: SEADS, proximity clouds, DSZ, and the dual extent method. All coding was done using Silicon Graphics (Silicon Graphics Indy 4600 PC, 100 MHZ, Entry graphics, 64 MB RAM, 4GB HD). All images are 512 $\times$ 512.

Four different scenes were used during performance testing: a) The first scene is called *Random* which is a scene with 2662 randomly sized and placed, colored spheres. This scene has very little spatial coherence. b) The second scene is called *Lanes* where 6400 small spheres are placed forming a cube of lanes. c) The third scene is called *Flakes* which has 1559 spheres in geometric formation. This scene has a lots of empty space in between the objects. d) The fourth scene is *Cars* where 1715 triangular patches are used.

For comparison with our results, it might be interesting to know that using when intersections are performed for all objects in the scene for every ray, the Random Scene 1 took 2 hours and 54 minutes to render.

Description of the table headers: *Scene:* Name of the scene. *Vw:* The view of the scene. *Preproc.:* Preprocessing time in seconds. *Rend.:* Rendering time in seconds. *Total:* Total execution time. *# of rays:* The total number of rays traced (x10^6). *# of trav.:* The total number of traversal steps (x10^6). *Avg. Trav.:* Average traversal distance including skips. *Ratio:* Ratio of the rendering time for the SEADS implementation and the method under consideration.

8 Analysis of Results

Tables 1-8, show a variety of empirical results. Table 1 shows the properties of each scene for three different grid sizes. Notice that the percentage of empty voxels is very high and therefore the *objects/voxel* column specifies the average number of objects considering only the occupied voxels. This gives us a hint about how many intersection tests are necessary every time the ray visits a non-empty voxel.

Memory requirements: A comparison of the memory requirements for each of the four methods

is in Tables 1 and 2. **Object memory:** The object memory is allocated during preprocessing to hold the pointers to objects. Every voxel containing objects has a linked list of object pointers. The requirements is therefore dependent on the number of objects in the scene and the level of subdivision. **Grid memory:** Every voxel data structure allocates enough memory to at least hold a pointer (4 bytes). If the voxel is empty, that is, if it does not contain any objects, the same memory space is used to hold distance maps used by the different skip methods. Depending on the method used, 4-7 bytes/voxel may be allocated.

We note that the object memory requirement of all the four methods is same. The grid memory requirement for DSZs and the Dual Extent implementations is twice of the SEADS and proximity clouds implementations.

9 Summary of the results

For every scene, we have three different viewpoints for that scene. Figure 8 and 9 show these scenes from a variety of angles. Tables 1-16 summarize our results.

As Tables 3-8 show, we have run exhaustive tests in comparing all the four methods using different grid sizes and scenes. The directed safe zone method also has a linear time algorithm for preprocessing the data with a larger constant of proportionality. This results in a slightly higher time for preprocessing for the DSZ method in comparison to the proximity cloud method. As expected, preprocessing times for the DSZ method is 2-3 higher that that of the proximity clouds. As preprocessing we need to find height fields for every slice in the grid, both the preprocessing time and memory requirements are higher for the dual extent method. For example, preprocessing times for the dual extents are 2-10 times higher in comparison to the proximity clouds. During preprocessing of the dual extent method, we exhausted the run time memory (RAM) available on our system for the grid size of 144 and the *Flakes* scene. That is the reason, statistics are not available for dual extent for the Flake scene in Tables 4 (Flake 1) and 8 (Flake 1-3). We note that the preprocessing is only necessary once for each scene which lets us render several images of different views and resolutions without calculating the distance maps. So, it is the rendering time which mainly determines the performance of an algorithm Following is the summary of our results from Tables 1-8:

- A large number of voxels are empty, and therefore it is beneficial to apply proximity clouds, DSZ or the dual extent method to data. In all cases, this is true as the value of the Ratio is always greater than one. The proximity, DE and DSZ methods are *effective* if we have empty cells. If we were using a small gridsize, the chances are that all cells may contain objects and therefore no *clouds* may form. In this extreme case, the DSZ, the DE and the proximity cloud method would fall back to a standard grid method in our implementation. In fact the DSZ, the DE and the proximity cloud algorithms induce some overhead, which makes standard traversal faster than DSZ skips for small cloud-distances ($D<5$) in our implementation (see also Section 3). Therefore, the reason for choosing a gridsize greater than 64 for our tests is to show the benefits of methods implemented in this paper.

- The DSZ will always be faster than the proximity cloud method, regardless of the gridsize, since the MIN() of all direction distances is the proximity cloud's distance value.

- For all the cases, Proximity Clouds, Directed Safe Zones, and the Dual Extent methods perform better than the SEADS implementation. This is supported by the fact that the value of Ratio in Tables 3-8 for Proximity clouds, DSZ and DE methods is always greater than 1.

- As the grid size increases (or voxels become smaller), the performance of the proximity clouds, DSZ or the dual extents method improve in their rendering times. Notice that the Ratio values increase with the grid size for these methods (Tables 3-8). Rendering times for the SEADS methods worsen because a larger percentage of time is invested passing through the empty regions.

- Rendering times for DSZ implementation are always equal or better that the proximity clouds method. Only exception to this is Lanes with grid size of 64 where proximity clouds is ever so slightly better than the DSZ method. However, as the grid size increases, DSZ becomes better than the proximity clouds for even the Lanes scene.

Proximity Clouds Implementation				
Scene Scene	Grid size Grid size	# of voxels	% of empty voxels	Objects per voxel
Random	62x64x62	246016	97.2	1.09
Random	125x128x125	2000000	99.0	1.03
Random	141x144x140	2842560	99.1	1.03
Lanes	63x63x64	254016	67.7	5.76
Lanes	126x127x128	2048256	75.2	3.66
Lanes	142x143x144	2924064	75.2	3.32
Flake	64x64x64	262144	94.7	1.59
Flake	128x128x128	2097152	97.0	1.15
Flake	144x144x144	2985984	97.2	1.16
Cars	64x24x14	21504	85.9	2.81
Cars	128x49x28	175616	93.0	1.84
Cars	144x55x31	245520	93.8	1.75

Table 1: Spatial subdivision statistics.

Proximity Clouds (DSZ and Dual Extents)			
Scene	Grid size	Object mem. in MBytes	Grid mem. in MBytes
Random	62x64x62	0.06 (0.06)	0.94 (1.88)
Random	125x128x125	0.16 (0.16)	7.63 (15.26)
Random	141x144x140	0.20 (0.20	10.84 (21.69)
Lanes	63x63x64	3.61 (3.61)	0.97 (1.94)
Lanes	126x127x128	14.20 (14.20)	7.81 (15.63)
Lanes	142x143x144	18.40 (18.40)	11.15 (22.31)
Flake	64x64x64	0.17 (0.17)	1.00 (2.00)
Flake	128x128x128	0.55 (0.55)	8.00 (16.00)
Flake	144x144x144	0.74 (0.74a)	11.39 (22.78*)
Cars	64x24x14	0.07 (0.07)	0.08 (0.16)
Cars	128x49x28	0.17 (0.17)	0.67 (1.34)
Cars	144x55x31	0.20 (0.20)	0.94 (1.87)

Table 2: Memory requirements: Proximity Clouds; DSZ and Dual Extent inside brackets. *a* means Not available for Dual extent.

- Most of the time, the Dual Extents method performs better than the proximity clouds (Tables 3-8).
- The Dual extents method performs better than both DSZ and proximity clouds method for the Lanes scenes because of the topology of the scene. This is the case for all the three grid sizes.
- For Random scenes, the directed safe zones method outperforms the proximity clouds by substantial amounts, as the grid size increases (See the Ratio values in Tables 3-8). Corresponding dual extent performances are in the middle of the DSZ and the proximity clouds methods.

10 Conclusions and Further Research

The Directed Safe Zones Method always performs better than the Proximity Clouds method because the skips are longer. For Random scenes the improvement of the rendering time is more than 25% compared to proximity clouds. We conclude that DSZ algorithm performs better than the proximity cloud method. In future, we plan to also run our algorithms on Eric Haines datasets [7].

Due to the nature of Dual Extents, it benefits from scenes forming lanes. This kind of topology is often found in architectural scenes. The Dual Extent Method proved to be 8-10% faster than even the DSZ method for scenes having a lot of rays parallel to the lanes. The skip operation of the Dual Extent Method uses a division operation in addition to the multiplications which slows it down, yet the rendering time is still lower for most scenes and grid resolutions compared to the Proximity Clouds Method.

We have shown that it is further possible to speed up ray tracing by taking advantage of spatial coherence of a scene by using the Directed Safe Zones and the Dual Extents method. We have successfully shown that it is possible to use directed distance maps to make the rendering even faster in comparison to the proximity clouds. As described in [15], further improvements over the DSZ method are possible when we use 24 directed (four per wall) values for voxels instead of six used in our implementation. In addition, we expect further gains in the rendering time when logarithmic distances are used.

Acknowledgments

We would like to thank the reviewers for their insightful comments on our paper.

References

[1] Whitted T., An improved illumination model for shaded display, *CACM*, 23(6), 343-349 (June 1980).

[2] Jevans D. and Wyvill B., Adaptive voxel subdivison for ray tracing, *Graphics Interface*, pp. 164-172 (June 1989).

[3] Goldsmith J. and Salmon J., Automatic Creation of Object Hierarchies for Ray Tracing, *IEEE CG&A*, pp. 14-20, May 1987.

[4] Kay T.L. and Kajiya J.T., Ray Tracing Complex Scenes, *Computer Graphics*, 20(4), pp. 269-278, SIGGRAPH Aug 1986.

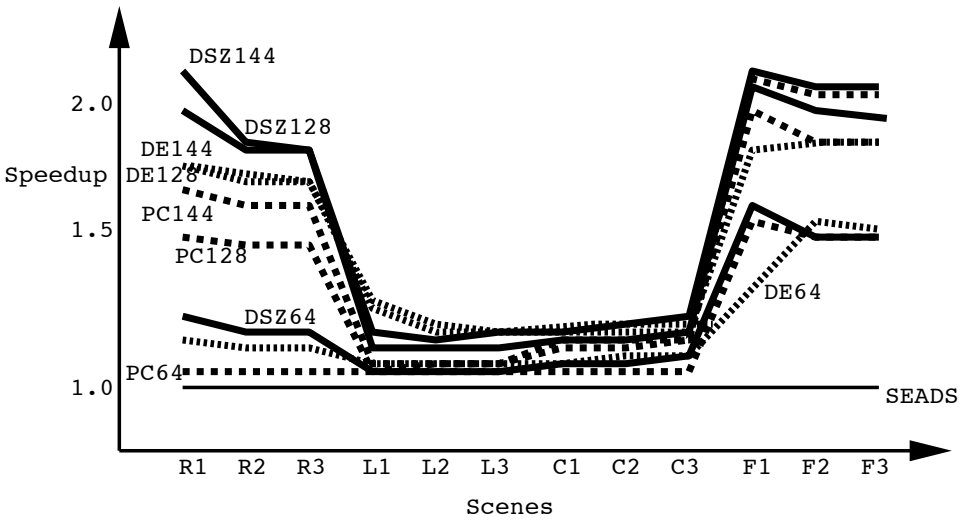


Figure 9: Speed up of all the four methods and grid sizes. R: Random, L: Lane, C: Car, and F: Flake.

[5] Arvo J. and Kirk D., Fast Ray Tracing by Ray Classification, *Computer Graphics*, 21(4), pp. 269-278, SIGGRAPH Aug 1987.

[6] Subramanian K.R. and Fussell Donald S., Automatic Termination Criteria for Ray Tracing Hierarchies, *Graphics Interface* , Calgary, Alberta, Oct. 3-7 (1991).

[7] Haines E.A., A Proposal for Standard Graphics Environments, *IEEE CG&A*, 7(11), pp. 3-5 (Nov 1987).

[8] Cleary J.G. and Wyvill G. Analysis of an algorithm for fast ray tracing using uniform space subdivision, *The Visual Computer 4*, pp. 65-83 (1988).

[9] Borgefors G. Distance transformation in digital images, *Computer Vision, Graphics and Image Processing*, 34, pp. 344-371 (1986).

[10] Glassner A.S. Space subdivision for fast ray tracing, *IEEE Computer Graphics And Applications*, 4(10), pp. 15-22 (1984).

[11] An Introduction to Ray Tracing edited by A.S. Glassner, Academic Press (1989).

[12] Fujimoto A, Tanaka T. and Iwata K. ARTS: Accelerated ray tracing system, *IEEE Computer Graphics And Applications*, 6(4), pp. 16-26 (1986).

[13] Cohen D. and Sheffer Z. Proximity clouds - an acceleration technique for 3D grid traversal, *The Visual Computer*, 11, pp. 27-38 (1994).

[14] Semwal S.K., Kearney C.K., and Moshell J.M. The Slicing Extent Technique for Ray Tracing: Isolating Sparse and Dense Areas, **IFIP Transactions**, vol. B-9, pp. 115-122 (1993).

[15] Kvanstrom H. The Dual extent and Directed Safe Zones techniques for ray tracing, MS Thesis, University of Colorado, Colorado Springs, pp. 1-72 (Spring 1996).

[16] Amantides J. and Woo A. A Fast Voxel Traversal Algorithm for Ray Tracing, *Proceedings EUROGRAPHICS*, pp. 3-10 (1987).

[17] Woo A. Fast Ray-Box Intersection, *Graphics Gems, I*, pp. 395-396 (1990).

[18] Cohen D. Voxel Traversal along a 3D Line, *Graphics Gems, IV*, pp. 366-368 (1994).

[19] Green D. and Hatch D. Fast Polygon-Cube Intersection Testing, *Graphics Gems, V*, pp. 375-379 (1995).

SEADS (Proximity clouds)							
Grid size: 64							
Scene-Vw	Preproc.	Rend.	Total	# of rays	# of trav.	Avg.trav	Ratio
Random-1	(6.98)	287.4 (270.9)	287.4 (277.9)	1.37 (1.37)	121.3 (96.6)	1 (1.25)	1 (1.06)
Random-2	(6.98)	260.3 (247.2)	260.3 (254.2)	1.37 (1.37)	105.6 (84.97)	1 (1.24)	1 (1.05)
Random-3	(6.98)	269.5 (255.7)	269.5 (262.7)	1.39 (1.38)	109.6 (88.15)	1 (1.24)	1 (1.05)
Lanes-1	(17.53)	273.6 (258.8)	273.6 (276.3)	1.65 (1.65)	40.91 (40.91)	1 (1)	1 (1.06)
Lanes-2	(17.53)	364.9 (341.4)	364.9 (358.9)	2.03 (2.03)	49.3 (49.3)	1 (1)	1(1.07)
Lanes-3	(17.53)	356 (330.1)	356 (347.6)	2.01 (2.01)	42.5 (42.5)	1 (1)	1 (1.08)
Flake-1	(7.67)	88.5 (58.1)	88.5 (65.7)	1.15 (1.15)	26.38 (6.21)	1 (3.69)	1(1.53)
Flake-2	(7.67)	83.72 (56.43)	83.72 (64.10)	1.15 (1.15)	23.81 (5.93)	1 (3.56)	1 (1.48)
Flake-3	(7.67)	83.84 (56.6)	83.84 (64.27)	1.15 (1.15)	23.77 (5.94)	1 (3.56)	1 (1.48)
Cars-1	(1.84)	77.73 (71.83)	77.73 (73.67)	1.21 (1.21)	8.69 (8.19)	1 (1.06)	1(1.08)
Cars-2	(1.84)	86.67 (79.97)	86.67 (81.82)	1.23 (1.24)	10.05 (9.38)	1 (1.07)	1 (1.08)
Cars-3	(1.84)	98.82 (91.18)	98.82 (93.02)	1.26 (1.26)	12.31 (11.37)	1 (1.08)	1 (1.08)

Table 3: Rendering statistics for SEADS, and the proximity clouds in brackets. Grid size: 64.

Directed Safe Zones (Dual Extents)							
Grid size: 64							
Scene-Vw	Preproc.	Rend.	Total	# of rays	# of trav.	Avg.trav	Ratio
Random-1	15.73 (223.1)	238.1 (250.70	253.83 (473.8)	1.37 (1.37)	64.56 (52.08)	1.86 (1.85)	1.21 (1.15)
Random-2	15.73 (223.1)	219.5 (233.3)	235.23 (456.4)	1.37 (1.37)	58.07 (47.35)	1.87 (1.81)	1.19 (1.12)
Random-3	15.73 (223.1)	227.3 (241.2)	243.03 (464.3)	1.39 (1.39)	60.73 (49.87)	1.86 (1.81)	1.19 (1.12)
Lanes-1	25 (36.9)	263.8 (250.2)	288.8 (287.1)	1.65 (1.65)	38.29 (30.61)	2.24 (1.27)	1.04 (1.09)
Lanes-2	25 (36.9)	346.1 (336.2)	371.1 (373.1)	2.04 (2.03)	45.33 (37.71)	2.34 (1.21)	1.05 (1.08)
Lanes-3	25 (36.9)	333.1 (329.7)	358.1 (366.6)	2.01 (2.01)	38.20 (33.69)	2.74 (1.15)	1.07 (1.08)
Flake-1	17.22 (n/a)	55.58 (n/a)	72.8 (n/a)	1.15 (n/a)	4.70 (n/a)	5.73 (n/a)	1.59 (n/a)
Flake-2	17.22 (1093)	54.44 (55.59)	71.66 (1148.59)	1.15 (1.15)	4.62 (4.11)	5.35 (5.20)	1.54 (1.51)
Flake-3	17.22 (1093)	54.57 (55.77)	71.79 (1148.77)	1.15 (1.15)	4.62 (4.14)	5.28 (5.19)	1.54 (1.50)
Cars-1	2.56 (7.82)	71.66 (70.67)	74.22 (78.49)	1.21 (1.21)	6.84 (5.63)	4.0855 (1.47)	1.08 (1.10)
Cars-2	2.56 (7.825)	79.45 (78.65)	82.01 (86.48)	1.23 (1.23)	7.63 (6.59)	4.3751 (1.50)	1.09 (1.10)
Cars-3	2.56 (7.83)	90.21 (90.43)	92.77 (98.23)	1.27 (1.27)	9.03 (8.38)	4.45 (1.39)	1.10 (1.09)

Table 4: Rendering statistics for Directed Safe Zones and the Dual Extent method in brackets. Grid size: 64.

SEADS (Proximity Clouds)							
Grid size: 128							
Scene-Vw	Preproc.	Rend.	Total	# of rays	# of trav.	Avg.trav.	Ratio
Random-1	(46.9)	482.9 (323.5)	482.9 (370.4)	1.37 (1.37)	243.57 (100.60)	1 (2.12)	1 (1.49)
Random-2	(46.9)	429.8 (297.2)	429.8 (344.1)	1.37 (1.37)	211.89 (91.13)	1 (2.07)	1 (1.45)
Random-3	(46.9)	445.2 (310.5)	445.2 (357.4)	1.389 (1.38)	219.92 (96.72)	1 (2.04)	1 (1.44)
Lanes-1	(74.17)	327.4 (311.4)	327.4 (385.57)	1.66 (1.66)	80.96 (67.98)	1 (1.15)	1 (1.05)
Lanes-2	(74.17)	422.4 (394.8)	422.4 (468.97)	2.04 (2.04)	96.82 (79.71)	1 (1.17)	1 (1.07)
Lanes-3	(74.17)	404.9 (376)	404.9 (450.17)	1.99 (1.99)	84.37 (68.81)	1 (1.18)	1 (1.08)
Flake-1	(49.98)	131.4 (66.39)	131.4 (116.37)	1.15 (1.15)	52.55 (6.99)	1 (6.49)	1 (1.98)
Flake-2	(49.98)	122.3 (64.86)	122.3 (114.84)	1.15 (1.15)	47.44 (6.79)	1 (6.17)	1 (1.89)
Flake-3	(49.98)	122.3 (65.08)	122.3 (115.06)	1.15 (1.15)	47.36 (6.82)	1 (6.14)	1 (1.88)
Cars-1	(6.59)	86.28 (76.84)	86.28 (83.44)	1.21 (1.21)	17.31 (11.79)	1 (1.45)	1 (1.12)
Cars-2	(6.59)	96.47 (85.67)	96.47 (92.2)	1.24 (1.24)	20.06 (13.77)	1 (1.42)	1 (1.13)
Cars-3	(6.595)	111 (98.51)	111 (105.12)	1.27 (1.27)	24.6 (17.13)	1 (1.42)	1 (1.13)

Table 5: Rendering statistics for SEADS, and the proximity cloud method in brackets. Grid size: 128.

Directed Safe Zones (Dual Extents)							
Grid size: 128							
Scene-Vw	Preproc.	Rend.	Total	# of rays	# of trav.	Avg.trav	Ratio
Random-1	118 (2856)	251.1 (284.7)	369.1 (3140.7)	1.37 (1.37)	53.99 (49.39)	3.59 (3.43)	1.92 (1.70)
Random-2	118 (2856)	232 (260)	350 (3116)	1.37 (1.37)	48.72 (43.85)	3.56 (3.45)	1.85 (1.65)
Random-3	118 (2856)	242.2 (273.1)	360.2 (3129.1)	1.38 (1.38)	51.96 (46.91)	3.47 (3.38)	1.84 (1.63)
Lanes-1	138.2 (743.6)	291.7 (266.8)	429.9 (1010.4)	1.65 (1.65)	47.21 (30.48)	2.11 (2.26)	1.12 (1.23)
Lanes-2	138.2 (743.6)	368.3 (357.3)	506.5 (1100.9)	2.03 (2.03)	53.55 (40.02)	2.22 (1.97)	1.15 (1.18)
Lanes-3	138.2 (743.6)	351.1 (354.4)	489.3 (1098)	1.99 (1.99)	45.13 (37.28)	2.41 (1.81)	1.15 (1.14)
Flake-1	127.4 (n/a)	64.56 (n/a)	191.96 (n/a)	1.15 (n/a)	5.48 (n/a)	9.23 (n/a)	2.04 (n/a)
Flake-2	127.4 (823.9)	63.51 (68.61)	190.91 (892.51)	1.15 (1.15)	5.44 (4.74)	8.52 (8.99)	1.93 (1.78)
Flake-3	127.4 (823.9)	63.72 (68.86)	191.12 (892.76)	1.15 (1.15)	5.47 (4.79)	8.46 (8.92)	1.92 (1.78)
Cars-1	12.72 (215.8)	74.3 (73.49)	87.02 (289.29)	1.21 (1.21)	8.83 (7.21)	3.77 (2.18)	1.16 (1.17)
Cars-2	12.72 (215.8)	82.08 (82.05)	94.8 (297.85)	1.23 (1.23)	10.04 (8.71)	3.53 (2.19)	1.18 (1.18)
Cars-3	12.72 (215.8)	93.08 (95.15)	105.8 (310.95)	1.26 (1.26)	12.01 (11.23)	3.32 (2.00)	1.19 (1.17)

Table 6: Rendering statistics for Directed Safe Zones, and the Dual Extent method in brackets. Grid size: 128. Data not available for Flake 1

SEADS (Proximity Clouds)							
Grid size: 144							
Scene-Vw	Preproc.	Rend.	Total	# of rays	# of trav.	Avg.trav.	Ratio
Random-1	(66.15)	532.4 (329.5)	532.4 (395.65)	1.37 (1.37)	273.88 (98.15)	1 (2.37)	1 (1.62)
Random-2	(66.15)	472.8 (303.4)	472.8 (369.55)	1.37 (1.37)	238.30 (89.12)	1 (2.31)	1 (1.56)
Random-3	(66.15)	489.9 (316.8)	489.9 (382.95)	1.38 (1.38)	247.35 (94.54)	1 (2.27)	1 (1.55)
Lanes-1	(103.3)	355.9 (334.7)	355.9 (438)	1.66 (1.66)	90.26 (70.8342)	1 (1.22)	1 (1.06)
Lanes-2	(103.3)	456 (421)	456 (524.3)	2.04 (2.04)	109.74 (83.9657)	1 (1.24)	1 (1.08)
Lanes-3	(103.3)	433 (398.4)	433 (501.7)	2.01 (2.01)	93.79 (71.42)	1 (1.24)	1 (1.09)
Flake-1	(70.4)	142.6 (70.43)	142.6 (140.83)	1.15 (1.15)	59.06 (7.24)	1 (7.04)	1 (2.03)
Flake-2	(70.4)	132.4 (68.89)	132.4 (139.29)	1.15 (1.15)	53.31 (7.05)	1 (6.68)	1 (1.92)
Flake-3	(70.4)	132.4 (69.05)	132.4 (139.45)	1.15 (1.15)	53.22 (7.08)	1 (6.656)	1 (1.92)
Cars-1	(8.631)	89.5 (78.62)	89.5 (87.25)	1.21 (1.21)	19.30 (12.32)	1 (1.51)	1 (1.14)
Cars-2	(8.631)	100.1 (87.64)	100.1 (96.27)	1.23 (1.238)	22.38 (14.46)	1 (1.50)	1 (1.14)
Cars-3	(8.631)	115.4 (100.9)	115.4 (109.53)	1.26 (1.26)	27.47 (18.06)	1 (1.49)	1 (1.14)

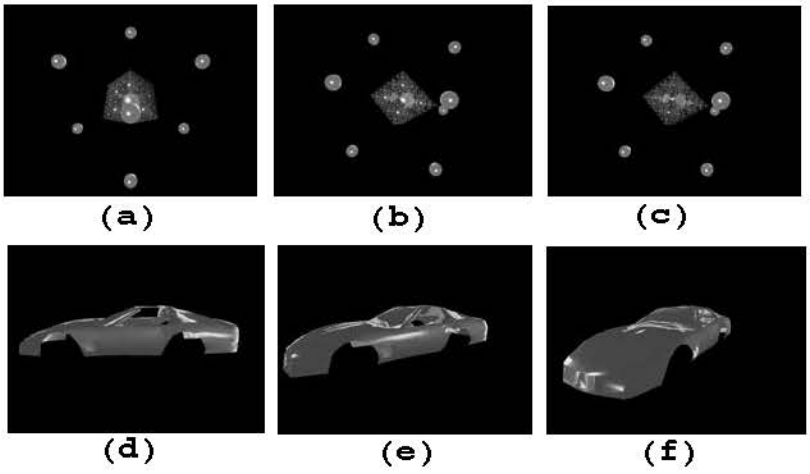
Table 7: Rendering statistics for SEADS, and the proximity cloud method in brackets. Grid size: 144.

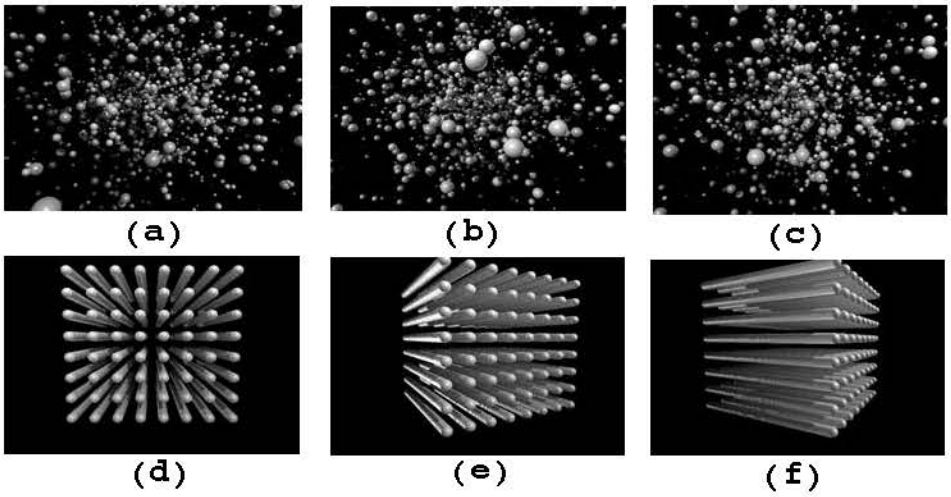
Directed Safe Zones (Dual Extent) Grid size: 144							
Scene-Vw	Preproc.	Rend.	Total	# of rays	# of trav.	Avg.trav	Ratio
Random-1	167.1 (1611)	254.1 (291.5)	421.2 (1902.5)	1.37 (1.37)	52.03 (48.70)	4.06 (3.84)	2.10 (1.83)
Random-2	167.1 (1611)	235.3 (266.3)	402.4 (1877.3)	1.37 (1.37)	47.01 (43.20)	4.01 (3.88)	2.01 (1.78)
Random-3	167.1 (1611)	245.4 (279.2)	412.5 (1890.2)	1.38 (1.38)	50.02 (46.12)	3.92 (3.80)	2.00 (1.76)
Lanes-1	194.6 (1316)	312 (286.2)	506.6 (1602.2)	1.66 (1.66)	48.32 (31.11)	2.17 (2.46)	1.14 (1.24)
Lanes-2	194.6 (1316)	390.2 (379.3)	584.8 (1695.3)	2.04 (2.03)	55.42 (41.63)	2.29 (2.13)	1.17 (1.20)
Lanes-3	194.6 (1316)	370.5 (375.4)	565.1 (1691.4)	2.01 (2.01)	46.36 (38.45)	2.48 (1.94)	1.17 (1.15)
Flake-1	181.1 (n/a)	68.77 (n/a)	249.87 (n/a)	1.15 (n/a)	5.65 (n/a)	9.99 (n/a)	2.07 (n/a)
Flake-2	181.1 (n/a)	67.74 (n/a)	248.84 (n/a)	1.15 (n/a)	5.63 (n/a)	9.20 (n/a)	1.96 (n/a)
Flake-3	181.1 (n/a)	67.92 (n/a)	249.02 (n/a)	1.15 (n/a)	5.67 (n/a)	9.13 (n/a)	1.95 (n/a)
Cars-1	17.23 (373.9)	75.76 (75.11)	92.99 (449.01)	1.21 (1.21)	9.16 (7.50)	3.75 (2.33)	1.18 (1.19)
Cars-2	17.23 (373.9)	83.54 (83.76)	100.77 (457.66)	1.23 (1.23)	10.43 (9.06)	3.55 (2.34)	1.20 (1.20)
Cars-3	17.23 (373.9)	94.72 (97.2)	111.95 (471.1)	1.26 (1.26)	12.50 (11.69)	3.33 (2.14)	1.21 (1.18)

Table 8: Rendering statistics for Directed Safe Zones, and the Dual Extent method in brackets. Grid size: 144.

Figure 10: Random (a, b and c), and Lanes (d, e, f) .

Figure 11: Flakes (a, b and c), and Cars (d, e, f) .


(a)
(b)
(c)
(d)
(e)
(f)


(a)
(b)
(c)
(d)
(e)
(f)