The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes

Michael Wand* Matthias Fischer† Ingmar Peter* Friedhelm Meyer auf der Heide† Wolfgang Straßer*

* WSI/GRIS, University of Tübingen † University of Paderborn

Abstract

We present a new output-sensitive rendering algorithm, the *randomized z-buffer algorithm*. It renders an image of an arbitrary three-dimensional scene consisting of triangular primitives by reconstruction from a dynamically chosen set of random surface sample points. This approach is independent of mesh connectivity and topology. The resulting rendering time grows only logarithmically with the numbers of triangles in the scene. We were able to render walkthroughs of scenes of up to 10^{14} triangles at interactive frame rates. Automatic identification of low detail scene components ensures that the rendering speed of the randomized z-buffer cannot drop below that of conventional z-buffer rendering. Experimental and analytical evidence is given that the image quality is comparable to that of common approaches like z-buffer rendering. The precomputed data structures employed by the randomized z-buffer allow for interactive dynamic updates of the scene. Their memory requirements grow only linearly with the number of triangles and allow for a scene graph based instantiation scheme to further reduce memory consumption.

Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation – Display Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques – Graphics data structures and data types; G.3 [Mathematics of Computing]: Probability and Statistics – Probabilistic algorithms.

Keywords: Rendering Systems, Level of Detail Algorithms, Monte Carlo Techniques

1 INTRODUCTION

Although the capabilities of computer graphics hardware have dramatically increased in the past years, the handling of scene complexity is still one of the most fundamental problems in computer graphics. Interactive display of highly complex scenes like landscapes with extensive vegetation, large city models, or highly detailed datasets in scientific visualization are a major challenge for rendering algorithms.

To be able to render highly complex scenes at interactive frame rates, the running time of the rendering algorithm has to be *output-sensitive*. Analogous to Sudarsky and Gotsman [26], we consider a rendering algorithm output-sensitive if its time complexity depends only weakly on the complexity of the input scene. Classic algorithms, like the z-buffer algorithm, fail to meet this requirement because their running times grow linearly with the number of elementary objects in the scene [26]. Although sophisticated hardware implementations are available, these algorithms are not capable of displaying highly complex scenes in real-time.

In this paper, we present a new output-sensitive rendering algorithm, the *randomized z-buffer algorithm*. The main idea of the algorithm is to generate an image of a scene by reconstructing it from a dynamically chosen set of random surface sample points. The sample points represent the complex scene geometry, hence not every single triangle must be handled separately during rendering. In a first step, random sample points are chosen so that they cover the projections of objects in the image plane approximately uniformly. This can be done in highly output-sensitive time. Here, the randomized selection is the key to avoid expensive geometric queries. In a second step, the algorithm reconstructs the occlusion between the chosen sample points and renders the resulting image using the visible points. Our approach has the following main advantages:

Efficiency: The rendering time for a scene consisting of n triangles covering an on-screen projected area of a pixels (including occluded triangles) is $O(a \cdot \log n)$. The logarithmic growth of the rendering time in n permits rendering of highly complex scenes. An automatic fallback strategy to conventional z-buffer rendering for low detail scene components insures that the rendering speed cannot drop below that of conventional z-buffer rendering. A caching strategy for sample sets is used to make optimal use of accelerated graphics hardware. Using a prototype implementation, we are able to render walkthroughs of scenes consisting of up to 10^{14} triangles at interactive framerates. The algorithm uses $O(n)$ storage and $O(n \cdot \log n)$ precomputation time. In order to store highly complex scenes in main memory, we adopted a scene graph based hierarchical instantiation scheme [29].

Generality: The algorithm takes arbitrary models consisting of triangles as input. Rendering times and results are independent of the input topology. Arbitrary local illumination models can be used to define the surface appearance.

Image quality: We will give analytical and experimental evidence that the image quality is comparable to conventional rendering results for a broad range of input scenes, covering most models found in practical applications.

Interactive editing: The data structures used for the selection of sample points allow for efficient dynamic updates: Insertion and removal of an object can be handled in $O(t)$ time where t is the height of an octree built for the objects in the scene. This permits interactive update times for local modifications of the scene.

The remaining part of this paper is organized as follows: The next section gives an overview on the randomized z-buffer algorithm. In Section 3, we briefly review related work. Afterwards, the two main steps of the algorithm, image reconstruction (Section 4) and sample point generation (Section 5), are discussed in detail. The image reconstruction is described first because it sets the preliminaries for the point selection scheme. In Section 6, the total running time of both steps is determined. Section 7 describes the automatic invocation of conventional z-buffer rendering, sample caching, and scene graph based scene encoding. Results are discussed in Section 8. The paper concludes with some ideas for future research.

* {wand, peter, strasser}@gris.uni-tuebingen.de
† {mafi, fmadh}@uni-paderborn.de

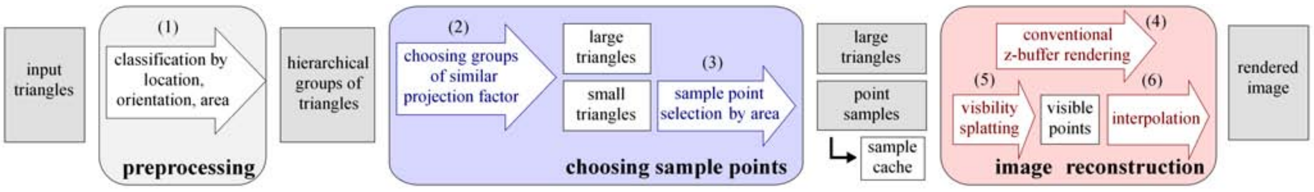


Figure 1: Outline of the algorithm. The randomized z-buffer dynamically chooses sample points from the scene (2, 3) using precomputed data structures (1). Afterwards, an image is reconstructed out of the sample points (5, 6). Low detail scene components are handled separately (4).

2 OUTLINE OF THE ALGORITHM

We assume that a scene $S \subseteq \mathbb{R}^3$ consisting of n triangles is given together with a shading function which determines the color for each point in S and which can be evaluated in O(1), independent of the scene complexity. The shading function may depend on the current viewpoint and global lighting settings. This includes e.g. Phong shading, texture and environment maps. For each frame, a planar perspective projection is given.

The randomized z-buffer rendering algorithm consists of two main steps: choosing sample points and image reconstruction (Figure 1).

Choosing sample points: An efficient algorithm for choosing sample points is the key to achieve interactive frame rates. We introduce a new randomized technique which efficiently generates a point sample representation of the scene on-the-fly, according to a given observer. To enable the correct reconstruction at a uniform resolution for all parts of the image, the sample points should be distributed uniformly on the projections of the objects in the image plane. Thus, for choosing sample points, a probability density proportional to the projected area of the surfaces of the objects has to be used. However, a sample selection using this viewpoint-dependent probability density does not seem to be possible in sublinear time. Therefore, we use an approximation algorithm. In a preprocessing step (Figure 1, (1)), the scene is subdivided into a hierarchy of groups of triangles which show a similar scaling factor during projection (*projection factor*). For each viewpoint, a suitable set of groups is chosen dynamically from the hierarchy (2). The maximum projection factor is estimated to determine the number of sample points to be chosen from each group: In most cases, it is sufficient to consider only the distance between a group and the viewer in order to estimate the projection factor. In special cases, better results can be achieved by additionally taking into account the orientation of the triangles. The sample points are then chosen within a specific group according to the unprojected area of the triangles (3) using precomputed search structures. We can show that this approximation does not harm the image quality and that the time for extracting groups from the hierarchy is short for "average" scenes, i.e. scenes with uniformly distributed surface normal vectors.

After a set of sample points has been chosen for a group of objects, they are stored in a *sample cache* to minimize sample selection costs. The sample points are reused for subsequent frames as long as the required sampling density does not change significantly. Since sampling of a triangle with large projected area turns out to be more expensive than rendering using the conventional z-buffer algorithm, we identify triangles with large projected area in our group selection scheme (1), (2) and render them using the conventional algorithm (4). Only triangles with small projected area will be replaced by sample points. This fallback strategy guarantees that the rendering speed cannot drop below that of the conventional z-buffer algorithm in any case.

Image reconstruction: To reconstruct an image from the sample points, the occluded points have to be removed from the sample set first (5). Afterwards, the image is obtained by interpolation between the visible sample points (6). Both can be done in one step by drawing the sample points into a z-buffer. If the sample density is high enough so that every pixel in the image receives a sample point from a foreground object the occlusion in the scene will be reconstructed correctly.

Using a visibility splatting approach similar to [18], the reconstruction can be adjusted both in quality and speed. Using larger splats of constant color and depth accelerates the reconstruction. High quality results can be obtained by weighted averaging of visible sample points using Gaussian filters. In this way, noise and aliasing artifacts can be avoided on the expense longer, non-real-time reconstruction times.

3 RELATED WORK

A large number of approaches have been proposed to handle high scene complexity in interactive applications:

Multi-resolution modeling: The objects of the scene are stored in different levels of detail (LOD). The appropriate LOD is chosen during rendering according to a view-dependent error metric. LODs can be generated automatically using *mesh-simplification*-algorithms (see e.g. Puppo et al. [20] for a survey). However, some scenes like scenes of complex topology cannot be simplified further without damaging the overall appearance. A forest consisting of many trees with hundreds of branches and leaves is an example of a scene which typically does not permit sufficient simplification.

Image-based rendering: Another approach to reduce the complexity of a scene is image based rendering. These techniques can be used to substitute fixed complexity representations derived from image data for objects of higher complexity. A large number of approaches were proposed which can be subdivided into two groups: The first group uses *precomputed* data structures, e.g. [24, 12, 15]. They differ from each other in generality, utilization of geometrical information, and memory requirements. A common limitation of all these methods is that the maximum image quality available is bounded by the resolution of the data, which is fixed in advance. Additionally, the illumination of the stored objects is fixed as well. In contrast to our approach, efficient dynamic scene modifications are not possible. The second group of image based data structures acquires their content *dynamically*, similar to the caching strategy used in our algorithm. In [25, 23], image caches are acquired by rendering parts of the scene using the conventional z-buffer algorithm. Thus, the cache update costs depend highly on the scene complexity. In contrast, the running time of our approach depends only logarithmically on the number of triangles in the scene.

Point sample rendering: Point sample approaches, like our algorithm, construct an image out of a set of surface sample points. This makes them independent of the input topology, overcoming the main problem of triangle mesh based multi-resolution models. The idea of using point samples to render surface models was first applied to the display of smooth three-dimensional surfaces by Levoy and Whitted [16], who already mentioned the possibility of using random sampling to obtain sample points. Earlier work focused on the display of non-solid objects like clouds, fire, or smoke [4, 7, 21]. Chamberlain et al. [5] render complex scenes using boxes of small projected size taken from a spatial hierarchy. The boxes are colored with averaged colors and transparency values of their content. Grossman and Dally [14] render complex object from point samples obtained from orthographic views of the object using a hierarchical push-pull algorithm for image reconstruction.

Two recent methods employ a multi-resolution point sample representation for rendering similar to our approach: Rusinkiewicz and Levoy [22] describe a rendering method for rapid display of large meshes from range scanner data dubbed *QSplat*. A bounding sphere hierarchy is built upon surface sample points storing average surface attributes such as normals and colors in every node. For rendering, the hierarchy is traversed until the projected size of a sphere falls below a certain threshold. Then splats are drawn on the screen with colors calculated from the averaged attributes and sizes according to the bounding spheres to avoid holes. Pfister et al. [18] converts the scene geometry into a point sample representation (*surfels*) by building a hierarchy of layered depth images [24]: For each node of an octree, point samples are generated by raytracing the objects in a fixed resolution from three orthogonal directions. For each sample point, position and surface attributes are stored. During rendering, an interpolation between adjacent levels in the octree is used to perform anti-aliasing.

A drawback of both methods is that the point sample representation used for display has to be built in a preprocessing step at a fixed resolution. The image quality available is then always bounded by the maximum sampling density of the point samples generated a priori. In order to obtain a good image quality even from close viewpoints, a large amount of memory must be spent to store a sampled representation of fine granularity. This is especially a problem if the level of detail varies across the scene as the sampling density is not adapted to the size of the underlying primitives. Thus, large triangles unnecessarily use up large amounts of memory.

Since our algorithm generates sample points on-the-fly from the original geometric scene description, the scene can be rendered from arbitrary viewpoints with constantly high quality using guaranteed O(n) storage for n triangles. Due to the selection overhead and the oversampling factor caused by the randomization, on-the-fly generation of sample points is slower than rendering precomputed point sets. We use a sample caching scheme to avoid this overhead: Sample sets with low oversampling ratio are constructed dynamically and can be reused for several frames. For views of a scene containing triangles with a large projected area, the automatic usage of conventional z-buffer rendering leads to an even better performance than purely point sample based rendering. In contrast to QSplat and Surfels our scene representation allows for efficient dynamic insertion and removal of objects. Thus, it can be used in a wider range of applications like interactive scene editing.

Raytracing: The raytracing algorithm [1] can be accelerated by a number of techniques to achieve rendering times with sublinear growth in respect to the number of objects in the scene. Commonly used methods are based on volume hierarchies, especially octrees, and regular grids [2]. Their effectiveness depends on the number of cells which have to be visited before a ray-object intersection is found. There are techniques from computational geometry, which guarantee to find intersections in logarithmic time (in respect to the number of objects) [8]. However, precomputation times and memory demands are prohibitive for large scenes. The randomized z-buffer circumvents the expensive inverse problem of finding ray intersections by placing sample points in object space. The disadvantage of this approach in comparison to raytracing is that the algorithm cannot resolve occlusions before rendering, which leads to a linear growth of the running time with the projected area (including occluded area).

Occlusion culling: Several algorithms have been proposed to avoid the handling of occluded parts of the scene by the rendering algorithm, see e.g. [13, 27, 30]. Our algorithm currently does not perform any occlusion culling, but view frustum culling only. We believe that occlusion culling and simplification, i.e. the removal of unnecessary details, are orthogonal problems and our algorithm concentrates on a solution to the simplification problem. However, the integration of many of the known occlusion culling techniques into our algorithm would be straightforward.

4 IMAGE RECONSTRUCTION

We start the description of the randomized z-buffer algorithm with the image reconstruction step. It is described first because it sets the preliminaries for the sample selection strategy depicted in Section 5.

We assume that a set of sample points is given which were chosen randomly, independently, and uniformly distributed on the projections of the objects in the image plane. This enables a reconstruction at a uniform level of detail for all parts of the image. We will describe and analyze the basic reconstruction technique in Section 4.1. Section 4.2 describes a generalization to a splatting algorithm which permits a more flexible trade-off between image quality and reconstruction speed.

4.1 Per-Pixel Reconstruction

The *per-pixel reconstruction* algorithm takes all sample points in arbitrary order, projects them onto the image plane, and draws them as pixels using a z-buffer to resolve occlusions.

To analyze the correctness of this method, we need a definition of a correct image first: We assume without loss of generality that every pixel is fully covered by triangles. An image is considered a *correct image of the scene* if every pixel of the image shows a color which can be found on any triangle fragment visible through that pixel. This definition of correctness merely abstracts from the slightly different behavior of rendering algorithms such as raytracing or z-buffering in handling sub-pixel details.

To obtain a correct image in the sense of this definition by per-pixel reconstruction, every pixel must receive at least one sample point from a foreground area. Additionally, one of the foreground points must have the smallest depth value among all sample points so that it will overwrite all other sample points in the z-buffer and produce a correctly colored pixel.

We now proceed in two steps. In the first step, we show how many sample points must be chosen so that every pixel receives at least one sample point from a foreground area. In the next step, we identify the conditions under which such a point will also be the point with the smallest depth value.

4.1.1 Number of Sample Points

In this section, the number of sample points which must be chosen to guarantee that every pixel receives at least one point from a non-occluded triangle fragment is determined.

First, we assume that there are no hidden surfaces in the image. As every pixel is by definition fully covered by visible surface fragments, the problem can be regarded as a simple urn model: Given v bins, how many balls must be thrown randomly, independently, with equal probability into the bins until every bin has received at least one ball? v is the number of “visible” pixels, i.e. the resolution of the image in pixels. Motwani et al. [17] shows that the expected value for the number of balls is $v{\cdot}H_v$ with H_v as v-th harmonic number, which is equal to $\ln v$ up to ±1. They show further that this result is asymptotically sharp so that one

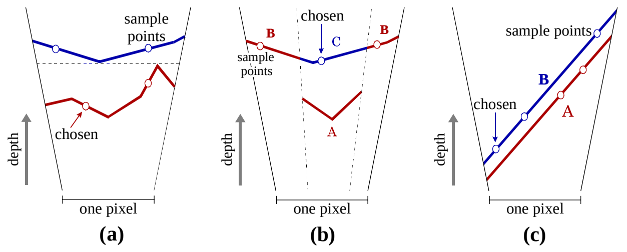


Figure 2: Different occlusion situations: (a) non-overlapping depth, (b) overlapping depth at object boundaries, (c) overlapping depth at parallel surfaces.

can conclude that O(v) additional balls are sufficient to fill every bin with arbitrary probability.

To take into account occlusions, we model the process as a two steps random experiment: Let h be the projected area of occluded surfaces measured in pixels and let $a := v + h$ be the total amount of projected area (these values will be estimated by the sample selection algorithm later on). The occluded area is now represented by h additional bins that are chosen with the same probability as the other v bins. Using a normal approximation to the binomial distribution, it can be shown that $a \cdot \ln v$ + O(a) sample points are sufficient to guarantee that every pixel receives at least one sample from a visible object fragment with arbitrarily high probability [28]. Experiments show that the absolute value of $a \cdot \ln v$ sample points is a good choice in practice. This delivers a sufficient sample density and decreasing the value by a small factor leads to "holes" in the surfaces. The oversampling factor $\ln v$ is caused by the randomization: For typical resolutions, on the average about 11-14 samples per pixel of projected area are necessary to guarantee that the projected surfaces are fully covered by sample points.

4.1.2 Correctness of the Image

Even if every pixel receives at least one sample point from a visible triangle fragment, in some cases pixels might show incorrect reconstructions. We distinguish two possible cases of sub-pixel occlusion:

1. *non-overlapping depth*: The maximum depth value of any visible triangle fragment is smaller than the minimum depth value of any occluded fragment in the pixel considered (Figure 2a). In this case, the point sample with the smallest depth value must stem from a visible triangle fragment and thus we always obtain a correct reconstruction.

2. *overlapping depth*: If the depth intervals of visible and hidden fragments overlap, a wrong reconstruction of occlusion is possible because the sample point with the smallest depth value might belong to an occluded triangle fragment. Such a point could incorrectly overwrite the sample points from visible areas. There are two typical cases: One possibility is that the pixel shows an object boundary or a sub-pixel detail in front of a background with higher depth. Figure 2b shows such an example: It is possible that sample points of region C overwrite all sample points of region B and no points from region A are available because it is only guaranteed that at least one sample is chosen from any visible triangle fragments of the pixel. Although formally the image can be incorrect, no significant loss of image quality is observed since this situation leads only to slightly "fuzzy" object boundaries.

Another typical situation is shown in Figure 2c: Two parallel surfaces, which are not perpendicular to the view direction, lie in very close proximity to each other. If the depth intervals of the surfaces are similar, the hidden surface B might shine through A with up to 50% probability since sample points from surface A and B are taken from nearly the same depth interval and show minimum depth with roughly equal probability. This problem can occur if offset polygons are used to model surface details. However, the problem can only be observed in practice if the relative distance of the surfaces is extremely small such that even a conventional z-buffer renderer tends to show occlusion artifacts due to its limited depth precision. The problem can easily be avoided by converting offset polygons into textures or removing the background underneath in the input model, as done in our example scenes. Thus, it is no restriction in practice.

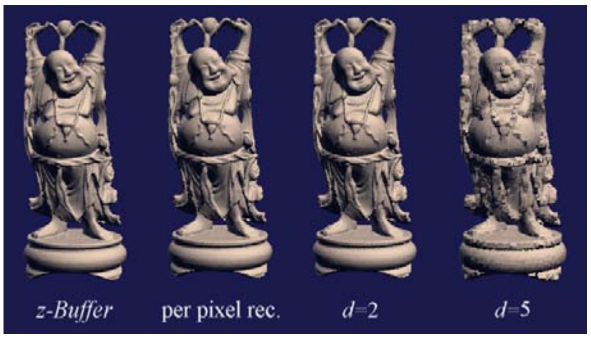


Figure 3: "Happy Buddha" mesh: Randomized z-buffer renderings with different splat sizes *d* in comparison to a conventional z-buffer rendering (leftmost): For small splat sizes the results are almost indistinguishable from the conventional z-buffer rendering.

4.2 Splatting

The per-pixel reconstruction algorithm presented in the preceding section is sometimes too expensive to run at interactive frame rates. A simple method for reducing the rendering costs is to use splatting: Instead of drawing single pixels, larger quadratic regions with side length d are filled with the same color and depth value, performing a depth comparison on each pixel. This reduces the size of the sample set necessary to safely cover all foreground area by $1/d^2$. Of course, the image quality drops off with increasing d (Figure 3). However, the results are always much better than those obtained by a simple reduction of the resolution by a factor d. We found that values of $d \approx 2..5$ are a good compromise. Using a value of $d = 2$ provides a good image quality even for scenes with high frequency details, saving about 75% computation time, while higher values may lead to objectionable visual artifacts.

To obtain reconstructions of higher quality we use splatting only to delete hidden sample points. Similar to [6], we then fill the image using a weighted average of adjacent visible points using Gaussian weighting functions (see [28] for details). We have to use a large sample size (about ten times the size for a per-pixel reconstruction) to avoid the low frequency noise artifacts which are emphasized by the filtering [11] so that this technique is not suitable for real-time applications.

5 EFFICIENT SAMPLE GENERATION

The objective of this section is to develop an algorithm for efficiently choosing sample points. The key observation for an efficient solution is that it is not necessary to guarantee an exactly uniform distribution of the sample points on the objects on the image plane. The image reconstruction will not fail if parts of the image contain *too many* sample points; these just cause additional processing costs. It must only be ensured that the sample density does not fall below the ideal sampling density anywhere. The running time of the reconstruction algorithm will then be increased proportional to the overestimation of the projected area.

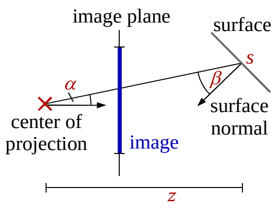


Figure 4: Perspective projection of a small surface fragment s.

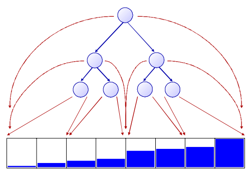


Figure 5: Hierarchical concatenation of distribution lists forming one global list.

We will now present such an approximation algorithm and show that the additional costs are small under reasonable assumptions about the scene geometry.

5.1 Probability Distribution

To obtain the probability density function for the selection of sample points, we consider an infinitesimal small surface fragment *s* and derive the *projection factor*, *prj*(*s*), by which it is scaled when it is projected on the image plane. The projection factor for such a surface fragment *s* is given by

$$prj(s) = \frac{1}{z^2} \cdot cut_{[0,1]}(\cos\beta) \cdot \frac{1}{\cos\alpha} .$$

z denotes the distance to the center of projection, measured orthogonal to the projection plane, β denotes the angle between the surface normal and a vector to the center of projection, and α denotes the angular deviation from the central view direction (Figure 4). $cut_{[0,1]}$ denotes clamping the value of the cosine to the interval [0,1]. This accounts for backface culling. As sampling density, we use the following probability density function *f*, which describes a uniform distribution on the projections of the surfaces on the image plane.

$$f(s) = \begin{cases} clip(s) \cdot prj(s) \Big/ \int_S clip(s) \cdot prj(x)dx, & \text{if } s \in S \\ 0, & \text{otherwise} \end{cases}$$

S denotes the set of all triangles of the scene. The function *clip* is defined to be 1 inside the current view frustum and 0 outside to limit the distribution of sample points to the area (potentially) visible from the current viewpoint. The view frustum is especially limited by a near clipping plane to exclude the singularity of the projection factor for $z = 0$.

5.2 Distribution Lists

We start the description of the sample selection process assuming that all projected area values of the triangles are known in advance. We choose random sample points according to the probability density function *f* in two steps: Firstly, a triangle is chosen with a probability proportional to its projected area. Secondly, a random point on the triangle is chosen using a random linear combination of its vertices.

A random triangle can be chosen efficiently using *distribution lists*: A distribution list is built by taking the triangles in arbitrary order and inserting a pointer to the triangle and an additional summed-area value per triangle into the list. We start with the projected area of the first triangle and increase the summed-area value by the projected area of each triangle inserted. In this way, we obtain a function which is proportional to the discrete cumulative distribution function for the probability for choosing the triangles.

To choose a random triangle, we determine a random number chosen uniformly from the interval [0,1] using a standard pseudo random number generator, scale the value by the total amount of projected area, and then search for the first entry in the list with a summed area value larger than the random value. This algorithm chooses the triangles according to the distribution function defined by the summed projected area values. It is a well known result from statistics [19] that the inverse of a distribution function φ applied to a random variable which is uniformly distributed in the interval [0, 1] creates a random variable which is distributed according to φ. With a binary search, the inversion of the discrete distribution function given by the distribution list can be performed efficiently. Using this technique, we can choose a sample point in O(log *n*) time for *n* triangles if the projected area values of all triangles are known. To avoid a costly computation of the projected area values, we substitute estimated values for the exact values in the next section.

5.3 Approximation

To estimate the projected area values, the scene is divided into groups of triangles showing a similar projection factor. The groups are obtained by choosing them dynamically from a precomputed hierarchy of triangle groups. For each selected group, the number of sample points it must receive is determined using the product of the unprojected area of all triangles in the group and the upper bound for the projection factor of that group. This guarantees that the sampling density does not fall below the minimum sample density required anywhere. For each group, a distribution list is stored according to the unprojected area of the triangles which is used to choose the sample points.

The division of the scene into groups must bound three factors since the total projection factor is a product of three independent terms: The *depth factor* $1/z^2$, the *orientation factor* $\cos\beta$, and the distortion towards the borders of the image $1/\cos\alpha$. The impact of the distortion on the projection factor is quite small. For typical values of the maximum view angle it can be ignored safely[1]. It remains to bound the two other factors.

5.3.1 Spatial Classification

To bound the depth factor, we build an octree for the triangles of the scene in a precomputation step. We can use *shortcuts* [3] to constrain the storage requirements of the octree to be linear in the number of triangles, i.e. all inner nodes, which have only one child node, are not stored explicitly. Every node has an *extended box*, which is the box of the node enlarged by a constant factor. Triangles, which intersect the octree grid, are stored in the smallest inner node which has an extended box that fully contains the triangle.

Each node in the octree holds a distribution list of the unprojected area values of all triangles contained in that node. For leave nodes, the lists are computed from the triangles themselves. Inner nodes obtain their list as a concatenation of the children's lists and the list of their own triangles in a fixed order. In total, only one list storing all summed area values is needed: The nodes mark up their part of the list by using pointers to their sections (Figure 5). Therefore, the overall storage requirement is only O(*n*) with *n* denoting the number of triangles and the precomputation can be performed in O(*n*·log *n*) time.

5.3.2 Dynamic Generation of Sample Points

When the observer moves to a new position, a set of octree boxes is chosen out of the octree. A *box selection algorithm* recursively traverses the octree from the root. The recursion selects all nodes intersecting the view frustum for which the ratio between the minimum and the maximum depth factor is not larger than a given constant ε (reasonable values for the *depth accuracy* ε are typi-

[1] The deviation caused by this distortion is also bound by the spatial subdivision used for bounding the depth factor such that even large view angles do not cause problems.

cally in the range of 1.1 - 2). It can be shown that during octree traversal only O(log τ) boxes are chosen in O(log τ) time where τ is the *relative depth range* of the scene. τ is defined to be the ratio between the smallest and the largest possible depth value in the scene. Thus, it is always bounded by the ratio of the distance of the front clipping plane to the diameter of the scene. The logarithmic growth of the number of boxes permits processing of very large scenes without significant costs for box processing. It can be shown further, that the cross-sectional area of an extended frustum enclosing the selected boxes exceeds that of the exact view frustum only by a constant [28].

5.3.3 Expected Deviation

Using the spatial hierarchy only to assemble groups of objects with similar projection factor already leads to a good approximation strategy. Recall that the increase of running time caused by a poor approximation of the ideal distribution is proportional to the overestimation of the projected area. To analyze the overestimation of the projected area due to the neglected orientation factor we assume that the normals of the triangles are distributed uniformly on the unit sphere. Under this assumption, it can be shown by simple integration [28] that the expected value for the overestimation of the projection factor due to the neglected orientation factor is 4. In the sample selection algorithm, surface sample points with backfacing normals can be detected at small costs. This reduces the overestimation of the orientation factor to 2. Therefore, one could expect an approximation accuracy and thus a runtime overhead of 2ε (with ε denoting the depth accuracy) for a scene with roughly uniformly distributed surface orientations. In a worst-case scenario, like a scene filled with triangles perpendicular to the view direction, an arbitrarily high overestimation can occur. However, such cases are rarely encountered in practice.

5.3.4 Classification by Orientation

In scenes with a non-uniform distribution of the surface normals we might use a more sophisticated classification scheme, which takes into account the orientation of the surfaces. We divide the unit sphere into classes of similar orientation by using a regular grid in polar coordinates. For each class of orientation, a spatial data structure is built as described in the preceding section. The additional information about the orientation of the objects is taken into account for the estimation of the projection factor.

Analytical as well as experimental results [28] show that this strategy improves the running time only in limited special cases with a very large projected area or a very low relative depth range. In other cases, especially in walkthrough applications of large environments with high relative depth range, this strategy appears to be counterproductive, leading to longer rendering times. Therefore, we use spatial classification only in our example renderings.

5.4 Dynamic Updates of the Scene

A dynamic version of the data structures for sample extraction can be obtained by applying some minor modifications to the static data structures. The main problem is that the distribution list storing the accumulated area values cannot be updated efficiently.

It is possible to substitute a dynamically balanced search tree for the distribution list, as shown in [28]. This leads to a dynamic update time of O(t) with t denoting the height of the spatial octree while preserving a sample selection time of O(log n) as in the static case. If the scene consists of objects which are uniformly distributed in at least one spatial dimension, the overhead of maintaining an additional dynamically balanced search tree can be avoided. For such scenes, the octree itself is relatively well balanced and it can be used as a "distribution tree": In each node the summed area value of all triangles of the subtree are stored. The sample selection algorithm now starts at the octree node which was selected by the box selection algorithm and generates a random number between zero and the summed area value of the last child node. Then it descends in the tree, choosing the last child node with a summed area value below the random value. Before descending further, the value is adapted to the range of the area values of the child node [28]. In this way, the correct leaf node can be found in O(t) time. Dynamic updates can be done in O(t) time as well: After adding or deleting a node from the octree in O(t) time an additional traversal from the leaf affected to the root node is necessary to correct the summed area values.

For practical applications, this variant is probably the more interesting one, because although using balanced search trees is asymptotically faster, they introduces a considerable overhead. Furthermore, the implementation of the second variant is much simpler and as the height of the octree is often not very large for practical scenes, the increase of the running time of the sample selection algorithm to O(t) may be acceptable. The experimental results (Section 8.4) confirm a good behavior in practice: For scenes with uniformly distributed objects, the octree based technique is nearly as fast as distribution lists for choosing sample points.

6 TOTAL RUNNING TIME

Summing up the running time for the image reconstruction and for choosing sample points, we obtain the following result:

Let S be a three dimensional scene consisting of n triangles with surface normals uniformly distributed on the unit sphere, a the projected area of the scene, v the resolution of the rendered image, and τ the ratio between the diameter of the scene and the minimum view distance. Then the scene can be rendered in $O(a \cdot \log v \cdot \log n + \log \tau) \subseteq O(a \cdot \log n)$ time using O(n) space for data structures which can be precomputed in $O(n \cdot \log n)$ time. Insertion and removal of triangles can be performed in O(t) time where t denotes the height of the octree build for the scene. In all pixels of the image which are fully covered by foreground objects and in which the depth intervals of foreground and background objects are disjoint, a correct image can be obtained with an arbitrarily high probability.

"Landscape scenes" are an important special case which can be roughly described as a disc with radius r in a plane on which objects are placed uniformly with limited height and uniformly distributed surface normal orientations. Typical examples are city scenes or natural landscapes. The projected area of such a scene grows logarithmically with r and the number of objects grows with r^2. If such a scene is rendered with the randomized z-buffer algorithm without any occlusion culling, we obtain a total running time of $O(\log^2 r)$. This is a big improvement in comparison to the conventional z-buffer algorithm, which would require $\Theta(r^2)$.

7 ENHANCEMENTS

7.1 Using Conventional z-Buffer Rendering for Large Triangles

For triangles which cover more than a small fraction of a pixel in the image it is not efficient to draw them using the random sampling algorithm. Instead it would be more efficient to use the conventional z-buffer algorithm. This can be faster by up to more than three orders of magnitudes for triangles with a large projected area.

To make use of this, all triangles which would receive more than a few sample points must be identified dynamically. The amount of sample points a triangle receives depends on the estimated projection factor (Section 5.3) and the unprojected area of the triangle. The estimated projection factor is known exactly, it is determined by the box selection algorithm. To estimate the second factor for groups of triangles, we use classes of triangles with a similar area. Each class holds triangles with area values which differ only by a constant factor from each other. Therefore, the size of the classes grows exponentially and O(log A) classes are necessary if A is the ratio between the smallest and the largest area value of a triangle in the scene. For each class, a separate data structure for the classification of the projection factor is built. The box selection algorithm described in Section 5.3.2 has to be modified: all selected boxes which contain triangles with a projected area smaller than a given threshold are scheduled for sampling. The triangles stored in the other boxes are passed to a conventional z-buffer renderer which uses the same depth- and framebuffer as the point sample renderer. Setting a conservative value for the threshold value ensures that the rendering speed never drops below that of a conventional z-buffer renderer.

The overhead caused by the introduction of multiple area classes is small. In contrast to classification by orientation, only a one-dimensional domain has to be subdivided. This is done in exponentially increasing spacing. Thus, the number of area classes is usually quite small. Experiments prove that the box processing times remain reasonably short (in practice, the number of boxes is enlarged by a factor of about 3-4).

7.2 Sample Caching

Our point sample rendering algorithm offers two particularly useful opportunities for optimizations: The binary search in the distribution lists causes a random memory access pattern unfavorable for modern microprocessor architectures. Additionally, the randomized point selection introduces an oversampling factor to the number of samples which grows with the natural logarithm of the display resolution (about 11-14 for typical resolutions). Utilizing temporal coherence, we can avoid both problems by applying a sample caching mechanism.

Before using sample points for image reconstruction, they are stored in a sample cache. In this way, they can be reused for several frames, as long as the sampling density does not change by more than a constant value. To remove unnecessary oversampling, the cached sample points are quantized to a regular three-dimensional grid in object space and duplicate points are removed. The grid spacing is chosen just small enough to safely cover all pixels on the screen, i.e. the maximum projected size of a grid block is $\sqrt{3}$ times the side length of a pixel (or splat, respectively). To minimize cache reconstruction efforts, multiple caches are used for regions of the scene in which the distance to the observer varies only by a constant value. This is realized by associating the caches with the octree boxes selected by the sample selection algorithm since these boxes always have this property. We do not use classification by orientation or backface culling so that the validity of a cached sample set depends only on the distance to the observer. Sample caches are deleted according to a least-recently-used algorithm, which guarantees fixed memory consumption.

Experiments show that the caching strategy can improve the rendering speed by an order of magnitude. The speedup depends on the movement of the observer: If the observer moves at a moderate speed, only a few cache builds are necessary and almost the full speedup is obtained. If the observer moves at a very high speed so that all geometry has to be resampled in each frame no speedup can be achieved. In practice, the caching scheme increases the rendering speed from a few seconds per image to interactive rendering times. In this way, the flexibility and low storage overhead of the random sampling algorithm can be combined with the high rendering speed of the deterministic methods like [18,22]. However, these provide the opportunity to apply expensive prefiltering techniques [18] to the point samples in the preprocessing step which cannot be accomplished on-the-fly.

7.3 Scene Encoding using Scene Graphs

Since the running time of the randomized z-buffer algorithm shows a very weak dependence on the number of triangles in the scene, the scene complexity which can be handled is several orders of magnitude higher than that of typical input scenes for conventional algorithms. Therefore, the main barrier for the display of highly detailed scenes is the memory consumption of the scene description. To be able to encode meaningful scenes of high complexity, we have adopted a scene graph based hierarchical instantiation scheme: The scene is described by a scene graph as found in many software libraries [29]. A separate octree is constructed for all nodes with multiple incoming references. If such a reference is inserted into the parent octree, a pointer to the separate octree of the referenced object is stored along with a transformation matrix instead of all triangles belonging to the object. This scheme is carried on hierarchically so that scenes of high complexity can be encoded.

As every instantiation layer requires one additional transformation of the sample point during the sample selection stage, the running time is increased by a constant value for each layer used. Experiments showed additional costs of 30% of the running time without instantiation per extra layer of instantiation for a pure software implementation.

8 IMPLEMENTATION AND RESULTS

To examine image quality and rendering performance of the randomized z-buffer in practice, we used a prototype implementation developed in C++ using OpenGL for low-level rendering. All results were obtained using an 800Mhz Athlon processor system with 512MB of main memory and an nVidia GeForce-2 GTS graphics board. The z-buffer renderer used for comparison was not optimized, in particular it did not make use of triangle strips in all possible cases. However, the implementation of the randomized z-buffer also leaves some room for optimization so that the comparison is still reasonable.

8.1 Scene Complexity

We used a chessboard explicitly modeled of black and white triangles as a test scene and increased the number of squares on the board to measure the dependence of the running time on the scene complexity. These settings ensured that the projected area and other geometric parameters remained constant for all levels of detail. In this way, the orthogonality of the results was preserved. The scene was rendered using point sample rendering only. Sample caching and the automatic invocation of conventional z-buffer rendering were disabled.

Figure 6 depicts the running time for an exponentially increasing scene complexity. The randomized z-buffer shows a roughly linear increase in running time proving the logarithmic growth with the number of triangles. The discontinuities show up at points where additional instantiation layers had to be inserted into the scene graph. Overall, the rendering time was only increased by about a factor of 4.6 between 2048 triangles and 10^{14} triangles and most of the costs are spent for transforming sample points in instances. This clearly shows the highly output-sensitive character of the running time. It is also apparent that pure random point sample rendering leads to a high overhead for low complexity scenes in comparison with conventional z-buffer rendering. This motivates the conventional z-buffer rendering of triangles with large projected area as described in Section 7.1.

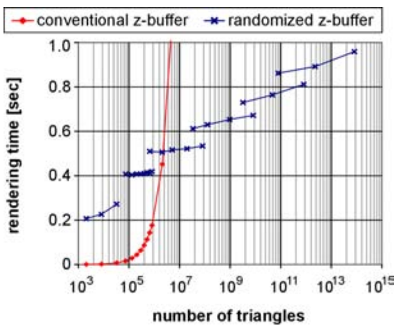


Figure 6: Running time dependent on the scene complexity. The rendering time of our algorithm grows roughly logarithmically with the scene complexity.

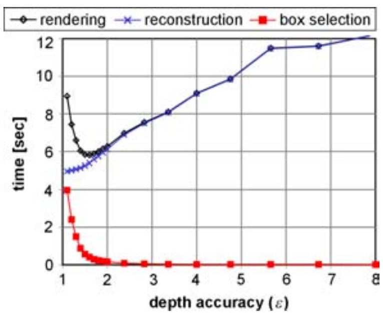


Figure 7: Trade-off between box selection and reconstruction time. The global minimum can be easily obtained since both box selection and reconstruction time are monotonic functions of the depth accuracy ε.

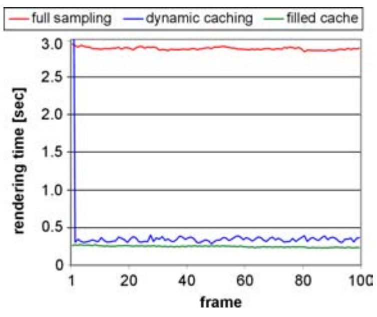


Figure 8: Rendering time for full rendering, sample caching and caching with filled caches: The fully dynamic sample caching strategy speeds up rendering by a factor of about 7.

8.2 Sampling Parameter Optimization

The main parameter which influences the point sample generation is the depth accuracy ε. Choosing the parameter ε is a trade-off between box selection time and image reconstruction time. It can be shown analytically [28] that on the one hand the number of boxes selected by the box selection algorithm grows rapidly with decreasing ε, by $\Theta((\sqrt{\varepsilon}-1)^{-3})$. On the other hand, the overestimation of the projected area increases with $\Theta(\sqrt{\varepsilon})$ for uniformly distributed objects. The sum of these two monotonic costs functions form an absolute minimum for some value ε_0. It depends on the projected area, which scales the reconstruction costs, and on the relative depth range τ, which scales the box selection costs.

Figure 7 depicts measured running times for an example scene, which confirm the theoretical analysis: The trade-off between the running times of the two steps of the randomized z-buffer algorithm is clearly visible. Furthermore, a strong growth of the total running time for small values of ε in contrast to a moderate growth for large values can be verified. For practical applications, it would be desirable to determine ε_0 automatically. As there is only one local minimum, this could be easily accomplished using a conventional minimum search algorithm.

8.3 Combination with Conventional z-Buffer Rendering

The combination of conventional z-buffer rendering with point sample rendering as described in Section 7.1 is controlled by two parameters: the maximum area ratio between the smallest and the largest triangle in an area class and the threshold value for the estimated projected area at which conventional z-buffer rendering is invoked. In our experiments, the maximum area ratio had little impact on the running time. Values of 2 to 64 led to similar results. The reason for this is that firstly, the accuracy of the estimation of the projected area has little influence on the running time as it affects only a small group of triangles with an area close to the threshold value. Secondly, the value for the area ratio does not affect the total number of boxes to be processed severely because the absolute number of area classes always remains quite small. The other parameter, the optimal value for the threshold at which conventional rendering is used, was approximately equal to the splat size divided by the oversampling ratio $d^2/\ln v$, as theoretically expected. It was independent of the scene properties and the camera position.

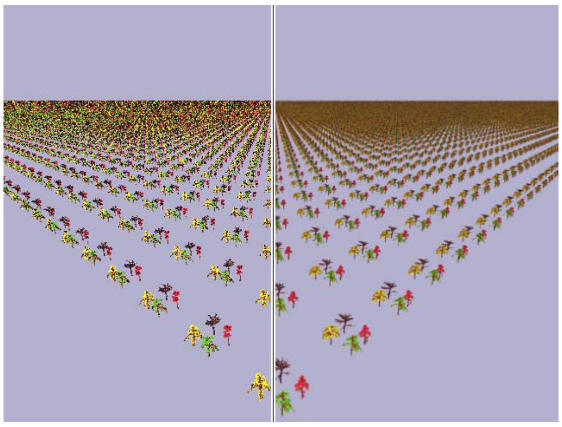
8.4 Dynamic Updates

We implemented the variant of the dynamic distribution tree which stores additional summed area values in the nodes of the octree as described in Section 5.4. To test dynamic scene editing we used the city scene from Section 8.6 replicated on a quadratic grid obtaining a scene consisting of 1.2 million triangles. We did not use any instantiations because dynamic operations are always performed within one of the separated octrees of an instance and the algorithm does not distinguish between triangle entries and instance entries. Thus, inserting or removing a complex object with a separate spatial data structure requires the same efforts as a single triangle. We inserted a model consisting of 5804 triangles into the scene triangle by triangle and removed it again. Rebuilding the octree from scratch took about 84 seconds in the static case while the dynamic update took only 570 msec (98 µsec/triangle). This allows for modifications of parts of the scene typical for interactive editing applications. Although in this implementation the distribution tree for sample selection is not guaranteed to be balanced optimally, we did not observe a loss of performance of more than 30% for any of our example scenes (in comparison to the static case). Hence, this turns out not to be a limitation in practice.

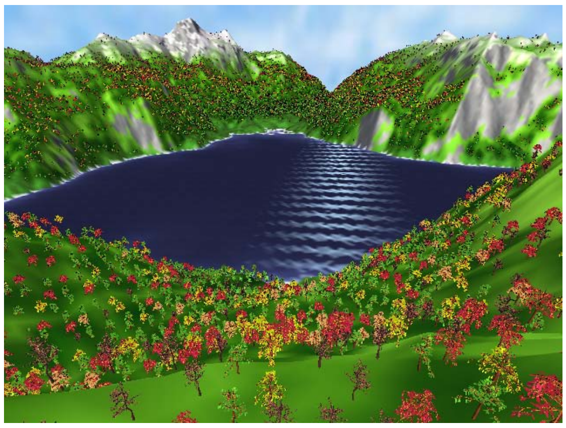
8.5 Sample Caching

If hardware accelerated geometric transformations are available, as on our test platform, the sample generation turns out to be the bottleneck of the algorithm. Thus, we obtain a significant speedup if cached sample sets can be used as described in Section 7.2. We applied the caching strategy to the example renderings of the next section: For high-resolution reconstruction, where the rendering costs are not dominated by box selection time and rendering of large triangles, rendering is speeded up by a factor of about 10-14. Quantization of the samples to a regular grid accounts for a factor of about 1.5 - 2 within this factor.

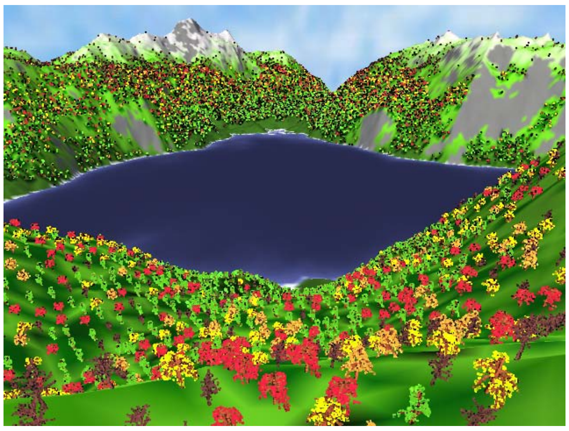
Figure 8 shows the rendering times during a walkthrough for an example scene with uniformly distributed tree models consisting of 9×10^8 triangles, similar to that shown in Figure 9b. We obtained an average speedup factor of 7.1 when sample caching was activated, starting with empty caches. The frame rate deviated from the average by ±15% due to cache update operations. If the same walkthrough is performed with precomputed cached sample sets, we obtain a speedup factor of 11.6, i.e. in this example, the cache updates account for 40% of the running time.



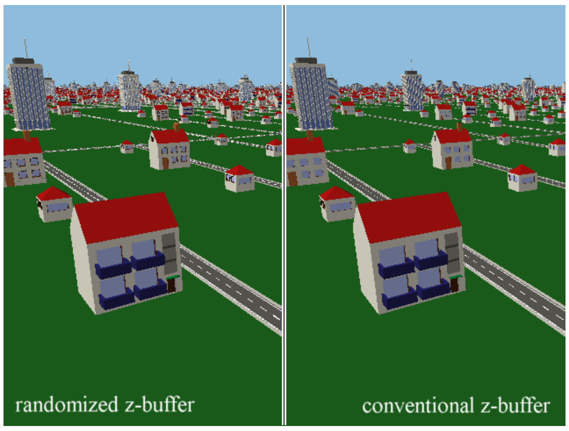
(a) Replicated trees ($1.4{\times}10^{14}$ triangles), left: per pixel reconstruction (9.1 sec without caching), right: Gaussian reconstruction (121 sec).



(b) Landscape scene ($4.1{\times}10^{8}$ triangles), per-pixel reconstruction ($d = 1$), Phong illumination model, rendering time: 19.2 sec (without caching)



(c) Landscape scene ($4.1{\times}10^{8}$ triangles), splatting, $d = 2$, diffuse illumination model, rendering time: 262 msec



(d) City model ($3.4{\times}10^{8}$ triangles), left: randomized z-buffer ($d = 1$), rendering time 422 msec, right: z-buffer, rendering time 8 min 43 sec

Figure 9: Example renderings, rendering times for (c), (d) were obtained using sample caching. See Table 1 for details.

8.6 Example Renderings

To show the benefits of the randomized z-buffer algorithm, we applied it to four different example scenes as shown in Figure 9. Table 1 summarizes the results.

Replicated trees: To demonstrate the ability of handling scenes of extreme complexity we used a scene with $5.9{\times}10^{9}$ instances of complex tree models consisting of about 23,300 triangles each replicated on a quadratic grid. A conventional z-buffer rendering would have taken about 160 days (assuming an optimistic triangle render rate of 10^{7} triangles per second). The randomized z-buffer produces good results after a few seconds. Figure 9a shows a per pixel reconstruction and a Gaussian reconstruction. Applying sample caching, the per-pixel reconstruction can be computed at nearly interactive frame rates (see Table 1). The anti-aliased reconstruction took two minutes to compute and shows no aliasing and only little noise artifacts. Especially the far field demonstrates the ability of random point sampling to produce a natural visual impression of a highly detailed geometry.

Landscape scene: A visually more appealing model consisting of about 410 million triangles was constructed by placing instances of the tree models on a fractal landscape (Figure 9c). It was rendered using sample caching in 71 msec (splatting, $d = 5$) to 410 msec (per pixel reconstruction, $d = 1$). For walkthroughs, the rendering times were moderately increased by a small offset, depending on the speed of movement. By replacing the static illumination model (which was used in all other examples as well) by a more expensive view dependent Phong model, a high quality rendering (Figure 9b) was obtained in 19 seconds. It shows no visually perceivable differences to a conventional z-buffer rendering which took about 9 minutes.

"Happy Buddha" mesh: Some approaches which perform well on unstructured scenes like forest scenes have severe problems on smooth meshes [5]. However, our algorithm delivers correct results in such cases, too: Figure 3 shows rendering results for the well-known "Happy Buddha" mesh (taken from [10]) in comparison to traditional z-buffer rendering. The image quality at small splat size ($d = 2$) is nearly indistinguishable from a conventional z-buffer rendering. Using sample caching, this result could be obtained in 30 msec, which is much faster than conventional z-buffer rendering. However, it should be noted that higher speed-ups can be achieved using mesh simplification (see e.g. [10]) since this scene, in contrast to our other examples, still contains a large amount of redundant triangles.

City scene: The city scene consists of a large amount of independent objects which are hard to handle for approaches like mesh-simplification [20]. The $3.4{\times}10^{8}$ triangles of the scene vary significantly in size so that automatic invocation of conventional z-buffer rendering for large triangles is particularly useful. A conventional z-buffer rendering took 8 minutes and 43 seconds. Using the randomized z-buffer we achieved up to 5-9 frames per second, depending on the splat size and the speed of movement of

Scene	complexity [# triangles]	memory consumption	precomp. time	Rendering time conv. z-buffer	rendering time (new sample set)			rendering time (sampling from cache)		
					$d = 1$	$d = 2$	$d = 5$	$d = 1$	$d = 2$	$d = 5$
“Happy Buddha”	1,085,634	163 MB	129 sec	1.18 sec	1.54 sec	0.44 sec	0.096 sec	0.11 sec	0.030 sec	0.007 sec
city scene	339,500,000	24,8 MB	4 sec	8 min 43 sec	5.0 sec	1.5 sec	0.52 sec	0.42 sec	0.19 sec	0.11 sec
landscape scene (diffuse)	410,501,184	137 MB	123 sec	7 min 51 sec	6.1 sec	1.8 sec	0.64 sec	0.41 sec	0.26 sec	0.071 sec
replicated trees	1.37×10^{14}	36 MB	21 sec	> 160 d (est.)	9.1 sec	4.6 sec	1.3 sec	0.80 sec	0.43 sec	0.33 sec

Table 1: rendering times for different scenes and different splat sizes *d* at a resolution of 640×480 pixels.

the observer, while still delivering a good image quality. Figure 9d compares conventional z-buffer rendering results to those of the randomized z-buffer. Note that in contrast to occlusion culling techniques the rendering speed does not drop if the scene is rendered from a higher point of view.

9 CONCLUSIONS AND FUTURE WORK

We presented a new output-sensitive rendering algorithm, the *randomized z-buffer algorithm.* Its rendering time grows only logarithmically with the number of triangles. Therefore, the algorithm can be applied to highly complex scenes. Experiments show that scenes consisting of several billion triangles can be explored at interactive framerates with an image quality comparable to that of conventional rendering methods. Our algorithm is applicable to more general scenes than mesh simplification-based approaches. It avoids the storage and preprocessing overhead of image-based rendering approaches. Our algorithm also improves on recent point sample based simplification algorithms [18, 22]: The randomized z-buffer algorithm guarantees a memory consumption linear in the number of triangles without restrictions to the sampling resolution, allows for efficient dynamic updates, and is never less efficient than conventional z-buffer rendering.

There are some possibilities for extending the methods described in this paper in future work:

Occlusion culling: It may be possible to make use of statistical information from the sampling process to cull occluded portions of the scene by detecting octree boxes which yield samples that are often rejected by the z-buffer test.

Modeling: A compact encoding is a key requirement for handling complex scenes. In principle the randomized sampling approach is not limited to triangular primitives. Thus, one should consider the usage of higher order primitives like parametric spline surfaces or simple generative models. Furthermore, it would be desirable to examine more sophisticated instantiation schemes. Subdivision in parameter space for the classification by projection factor and parametric distortion could serve as a basic technique to implement enhanced modeling techniques.

Acknowledgements

The authors wish to thank Michael Doggett, Tamás Lukovszki, Christian Sohler and Stanislav Stoev for valuable discussion and their support of our work. This work was partially supported by DFG grant Me872/7-3 and grant Str465/2-1(part of V^3D^2) and the IST Program of the EU under contract number IST-1999-14186 (ALCOM-FT). Part of this work was done while Michael Wand was affiliated with the Department of Mathematics & Computer Science and Heinz Nixdorf Institute of the University of Paderborn.

References

[1] Appel, A.: Some Techniques for Shading Mashine Renderings of Solids. In: *Proceedings of the Spring Joint Computer Conference,* 37-45, 1968.

[2] Arvo, J., Kirk, D. A: Survey of Ray Tracing Acceleration Techniques.In: *Glassner, A. (editor): An Introduction to Ray Tracing, Academic Press, 4th printing,* pp. 201-262, 1991.

[3] Bern, M., Eppstein, D., Gilbert, J.: Provably good mesh generation. In: *Proc. 31st Annu. IEEE Sympos. Found. Compt. Sci.,* 231-241, 1990.

[4] Blinn, J. F.: Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. In: *Computer Graphics (SIGGRAPH 82 Proceedings), 16 (3), 21-29,* 1982.

[5] Chamberlain, B., DeRose, T., Lischinski, D., Salesin, D., Snyder, J.: Fast Rendering of Complex Environments Using a Spatial Hierarchy. In: *Proc. Graphics Interface ’96,* 132-141, 1996.

[6] Cook, R.L.: Stochastik Sampling and Distributed Ray Tracing. In: Glassner, A. (editor): *An Introduction to Ray Tracing.* Academic Press, 4th printing, pp. 161-199, 1991.

[7] Csuri, C., Hackathorn, R., Parent, R., Carlson, W., Howard, M.: Towards an Interactive High Visual Complexity Animation System. In: *Computer Graphics (SIGGRAPH 79 Proceedings), 13 (3), 289-299,* 1979.

[8] de Berg, M., Halperin, D., Overmars, M., Snoeyink, J., van Kreveld, M.: Efficient Ray Shooting and Hidden Surface Removal. In: *Algorithmica,* 12, 30-53, 1994.

[9] Feller, W: *An Introduction to Probability Theory and Its Applications.* Third Edition, revised printing, Wiley & Sons, 1970.

[10] Garland, M.: *Quadric-Based Polygonal Surface Simplification,* Ph.D. thesis, Technical Report CMU-CS-99-105, Carnegie Mellon University, 1999

[11] Glassner, A. S.: *Principles of Digital Image Synthesis.* Morgen Kaufmann Publishers, 1995.

[12] Gortler, S. J., Grzeszczuk, R., Szeliski, R., Cohen, M. F.: The Lumigraph. In: *SIGGRAPH 96 Proceedings, Annual Conference Series, 43-54,* 1996.

[13] Green, N., Kass, M., Miller, G.: Hierarchical Z-Buffer Visibility. In: *SIGGRAPH 93 Proceedings, Annual Conference Series,* 231-238, 1993.

[14] Grossman, J. P., Dally, W.: Point Sample Rendering. In *Rendering Techniques’98,* 181–192, Springer, 1998.

[15] Levoy, M., Hanrahan, P.: Light Field Rendering. In: *SIGGRAPH 96 Proceedings, Annual Conference Series, 31-42,* 1996.

[16] Levoy, M., Whitted, T.: *The Use of Points as a Display Primitive.* Technical report, University of Norh Carolina at Chapel Hill, 1985.

[17] Motwani, R., Raghavan, P.: *Randomized Algorithms.* Cambridge University Press, 1995.

[18] Pfister, H., Zwicker, M., van Baar, J., Gross, M.: Surfels: Surface Elements as Rendering Primitives. In: *SIGGRAPH 2000 Proceedings, Annual Conference Series, 335-342,* 2000.

[19] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical Recipes in C, The Art of Scientific Computing,* Second Edition, Cambridge University Press, 1992.

[20] Puppo, E., Scopigno, R.: Simplification, LOD and Multiresolution Principals and Applications. In: *EUROGRAPHICS 97 Tutorial Notes,* 1997.

[21] Reeves, W. T.: Particle Systems – A Technique for Modeling a Class of Fuzzy Objects. In: *Computer Graphics (SIGGRAPH 83 Proceedings), 17 (3), 359-376, 1983.*

[22] Rusinkiewicz, S., Levoy, M.: Qsplat: A Multiresolution Point Rendering System for Large Meshes. In: *SIGGRAPH 2000 Proceedings, Annual Conference Series, 343-352,* 2000.

[23] Schaufler, G.: Per-Object Image Warping with Layered Impostors. In: *Rendering Techniques '98,* 145-156, Springer, 1998.

[24] Shade, J., Gortler, S., He, L., Szeliski, R.: Layered Depth Images. In: *SIGGRAPH 98 Proceedings, Annual Conference Series, 231-242,* 1998.

[25] Shade, J., Lischinski, D., Salesin, D. H., DeRose, T., Snyder, J.**:** Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In: *SIGGRAPH 96 Proceedings, Annual Conference Series, 75-82,* 1996.

[26] Sudarsky, O., Gotsman, C.: Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. In: *Computer Graphics Forum (EUROGRAPHICS 96 Proceedings), 15 (3), 249-258,* 1996.

[27] Teller, S.J., Séquin, C.H.: Visibility Preprocessing For Interactive Walkthroughs. In: *Computer Graphics (SIGGRAPH 91 Proceedings), 25 (4),* 61-69, 1991.

[28] Wand, M., Fischer, M., Meyer auf der Heide, F.: Randomized Point Sampling for Output-Sensitive Rendering of Complex Dynamic Scenes. technical report WSI-2000-20, WSI/GRIS University of Tübingen, 2000. http://www.gris.uni-tuebingen.de/publics/paper/Wand-2000-Randomized.pdf

[29] Wernecke, J.: *The Inventor Mentor: Programming Object-Oriented 3d Graphics With Open Inventor, Release 2.* Addison Wesley, 1994.

[30] Zhang, H., Manocha, D., Hudson, T., Hoff, K.: Visibility Culling using Hierarchical Occlusion Maps. In: *SIGGRAPH 97 Proceedings, Annual Conference Series, 77-88,* 1997.