Randomized Point Sampling for Output-Sensitive Rendering of Complex Dynamic Scenes

Michael Wand*

Tübingen University
Auf der Morgenstelle 10 / C9
D-72076 Tübingen, Germany

Email: miwand@gris.uni-tuebingen.de
WWW: http://www.gris.uni-tuebingen.de

Matthias Fischer†
Friedhelm Meyer auf der Heide

Heinz Nixdorf Institute
Paderborn University
Department of Mathematic and Computer Science
D-33095 Paderborn, Germany

Email: {mafi, fmadh}@uni-paderborn.de
WWW: http://www.uni-paderborn.de/cs/ag-madh/

Abstract

We present a new output-sensitive rendering algorithm, the randomized z-buffer algorithm. It renders an image of a three dimensional scene of triangular primitives by reconstruction from a random sample of surface points which are chosen with a probability proportional to the projected area of the objects. The approach is independent of mesh connectivity and topology. It leads to a rendering time that grows only logarithmically with the numbers of triangles in the scene and to linear memory consumption, thus allowing walkthroughs of scenes of extreme complexity. We consider different methods for image reconstruction which aim at correctness, rendering speed and image quality and we develop an efficient data structure for sample extraction in output-sensitive time which allows for efficient dynamic updates of the scene. Experiments confirm that scenes consisting of some hundred billion triangles can be rendered within seconds with an image quality comparable to a conventional z-buffer rendering; in special cases, realtime performance can be achieved.

---

* Part of the work was done while Michael Wand was affiliated with the Department of Mathematics & Computer Science and Heinz Nixdorf Institute of the University of Paderborn.

† Partially supported by DFG Grant Me872/7-3 and the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

Contents

1 Introduction

Interactive walkthroughs of large three-dimensional scenes are a major challenge for rendering algorithms. The ability to display complex scenes at interactive frame rates is one of the key requirements for many virtual reality applications. To be able to render views of scenes of increasing complexity in reasonably short time, the rendering algorithm must have output-sensitive complexity characteristics. Analogous to Sudarsky and Gotsman [31] we consider a rendering algorithm as output-sensitive if its time complexity depends only weakly on the complexity of the input scene. In addition, we allow the algorithm to produce approximate results, so its rendering time can also depend on the image quality achieved. Classical algorithms, like the z-buffer algorithm [5], fail to meet these requirements because their running times grow linearly with the number of elementary objects in the scene [31]. Although sophisticated hardware implementations are available, these algorithms are not capable of displaying complex scenes in realtime. Many improvements have been suggested to overcome these limitations, as summarized in the next section.

We present the *randomized z-buffer algorithm*, a new output-sensitive rendering algorithm. The main idea of the algorithm is to generate an image by reconstruction out of a sample of randomly chosen surface points. In the first step, the algorithm chooses random sample points on the surfaces of the objects with a probability roughly proportional to the projected area of the objects, i.e. the sample points are approximately uniformly distributed on the projections of the objects in the image plane. Efficiently choosing sample points is the key for output-sensitive rendering time. The selection of the sample points is performed approximately using a hierarchical scheme for classification of the triangles by spatial location and orientation. In the second step, the rendering algorithm reconstructs the occlusion between the sample points and constructs an image of the scene out of the visible points. The reconstruction process offers a trade-off between efficiency (splatting) and image quality (Gaussian filtering). The main advantages of the randomized z-buffer algorithm are:

Generality: The algorithm takes arbitrary models consisting of triangles as input, rendering times and results are independent of the input topology. Like the conventional z-buffer algorithm, any efficient local illumination model can be used to define the surface appearance.

Efficiency: It will be shown that the rendering time for scenes with uniformly distributed surface normal orientations can be bound to $O((a \cdot \log v + \log \tau) \cdot \log n)$ where a is the projected area (including occluded area) of the scene, v the resolution of the rendered image, τ the ratio between the diameter of the scene and the minimal viewing distance and n the number of triangles in the scene. The precomputed data structures use only $O(n)$ storage. 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. The logarithmic growth of the rendering time in n permits the rendering of extremely detailed scenes. Using a prototype implementation, we were able to render scenes of 10^{14} triangles within a few seconds. To be able to store scenes of that size in main memory, we adapted the data structures to support hierarchical instantiation to encode complex scenes using scene graphs [33].

Image quality: It can be shown that good image quality is achieved with arbitrary high probability under mild assumptions on the input geometry. We will give analytical and experimen-

tal evidences that the image quality is comparable to conventional rendering results for a broad range of input scenes, covering most models found in practical applications.

Suitable for interactive editing: The precomputed data structures for the selection of sample points allow for efficient dynamic updates: Insertions and deletions triangles and object instances can be handled in $O(\log n + h)$ time where h is the height of an octree built for the objects in the scene. This leads to interactive update times for local scene modifications.

2 Related Work

The handling of complexity is one of the most fundamental problems in computer graphics. Several approaches have been proposed to speed up rendering of complex scenes:

Multi-resolution modeling: The objects of the scene are stored in different levels of detail that are chosen for rendering according to an appropriate error metric taking into account the position of the observer and object properties. There is a number of "mesh-simplification"-algorithms which automatically generate differently detailed models from a fully detailed triangle mesh, see Klein [18], Heckbert and Garland [17] or Puppo and Scopigno [26] for a survey. A general problem of multi-resolution models is that scenes consisting of many independent objects or of objects of complex topology often cannot be simplified into a triangle mesh with only a few triangles. Scenes like a forest consisting of hundreds of trees with hundreds and thousands of branches and leaves each are examples of scenes which typically do not permit sufficient simplification.

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 and thus they overcome 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 [20]. Earlier work focused on the display of non-solid objects like clouds, fire or smoke [4, 9, 27]. Two recent methods employ *points samples* as rendering primitives in a fashion similar to our approach: Rusinkiewicz and Levoy [28] describe a rendering method dubbed *QSplat* for rapid display of large meshes from range scanner data. A bounding sphere hierarchy is built upon the mesh storing average surface attributes such as normals and color in every node. For rendering, the hierarchy is traversed and traversal is stopped when the projected size of a sphere falls below a certain threshold. Then a splat is drawn on the screen with a color calculated from the averaged attributes and a size proportional to the bounding sphere to avoid holes. Pfister et al. [24] a priori convert the scene geometry into a point sample representation ("*surfels*") by building a hierarchy of layered depth images [29]: For each cell 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. At rendering time, an interpolation between adjacent levels in the octree is used to perform anti-aliasing.

These methods operate fully deterministic, which leads to a higher performance than that of our approach because a random selection of sample points leads to higher oversampling. But the randomization provides some advantages: The averaged normals and color attributes used in *QSplat* cannot represent object clusters with local occlusions or surfaces with highly vary-

ing surface normals, e.g. rough surfaces or surfaces with small holes. In contrast, the random sampling is able to represent these phenomena over a wider range of cases on the expense of a higher running time, which is proportional to the local depth complexity. Both approaches need to convert all geometry a priori into a point sample representation while our algorithm dynamically chooses sample points from the scene. This leads to linear memory consumption, which cannot be guaranteed by presampling (e.g. if the scene contains triangles of strongly varying size) and a better image quality for viewpoints in close proximity to some objects: The resolution of the image is not limited by the maximum sampling frequency used for a priori sample generation. Additionally, our data structures allow for efficient dynamic insertions and deletions of objects.

Image based rendering: Image based rendering techniques can be used to substitute fixed complexity representations for objects of higher complexity. Early methods made use of textures as substitution for clusters of objects. The textures can be either precomputed [21] or generated dynamically [30]. More recently, refined rendering techniques have been developed, which can represent objects over a broader range of viewing angles, e.g. [15, 19, 11]. These methods offer a great amount of flexibility for modeling objects, including construction out of photographs of real world objects. Image based methods facilitate the representation of arbitrary complex models with a fixed complexity representation but a general disadvantage of the image-based approach is that storage requirements are often high because large amounts of image data has to be stored. This inhibits the rendering of large scenes, i.e. scenes consisting of many objects, each stored at high levels of detail, and leads to large costs for dynamic updates of the scene. Our approach constructs a low complexity representation directly out of the scene geometry on the fly using a simple octree-like data structure, so storage requirements and dynamic update times are small.

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 that have to be visited before a ray-object intersection is found. For a ray shot through a forest scene consisting of many small objects, missing most objects in close proximity the efforts to find an intersection can be large. There are techniques from computational geometry which guarantee to find intersections in logarithmic time [10] (in respect to the number of objects), but 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.

Occlusion culling: Several algorithms have been proposed to avoid handling occluded parts of the scene by the rendering algorithm [16, 32, 34]. Our algorithm currently does not perform any occlusion culling but only viewing frustum culling, leading to a rendering time that grows linearly with the projected area of the objects in the scene, including occluded area. 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 latter problem. Therefore, we will use benchmark scenes with low depth complexity to show the effective-

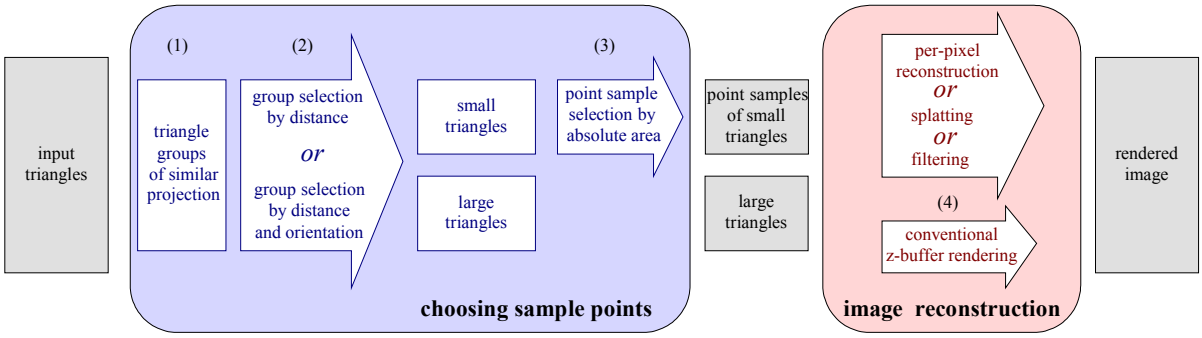


Figure 1: outline of the algorithm

ness of our approach. The randomized z-buffer algorithm offers opportunities for the integration of occlusion culling; this could be subject of future work.

3 Overview

We assume that a scene $S \subseteq \Re^3$ consisting of n triangles $t_1 ... t_n$ 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, so this includes e.g. phong shading, texture and environment maps. For each frame, a planar perspective projection is given to describe the camera settings.

3.1 Outline of the Algorithm

Our rendering algorithm consists of two main steps: choosing sample points and image reconstruction (see figure 1). In the first step, our goal is to choose sample points on the triangles with a probability proportional to their projected area. This leads to a uniform distribution of sample points on the projections of the objects of the scene in the image plane. In the second step, we investigate three different reconstruction algorithms to remove points from occluded surfaces and to fill a raster image according to the visible sample points.

Choosing sample points: An efficient algorithm for choosing sample points is the key to achieving output-sensitive rendering times. An exact solution which chooses sample points with a probability proportional to the projected area does not seem to be possible in sublinear time. Therefore, we use an approximation algorithm (see figure 1). It dynamically chooses groups of objects with a similar projection factor (1) from a precomputed hierarchy of groups and then chooses random sample points according to the absolute (unprojected) area of the triangles in the group using a binary search in lists of summed area values (3). This can be done in a time proportional to the logarithm of the numbers of triangles in the group. We investigate two different methods for the group selection (2): A selection of triangles by distance only and a selection by distance and orientation of the triangles. The spatial classification is given by an octree and the classification by orientation applies this to groups of triangles with similar normal vectors. We show that the approximation does not harm the image quality and that the time for extracting groups from the hierarchy is small for "average"

scenes, i.e. scenes with uniformly distributed surface normal vectors. However, the worst-case efforts of the approximation algorithm are unbound.

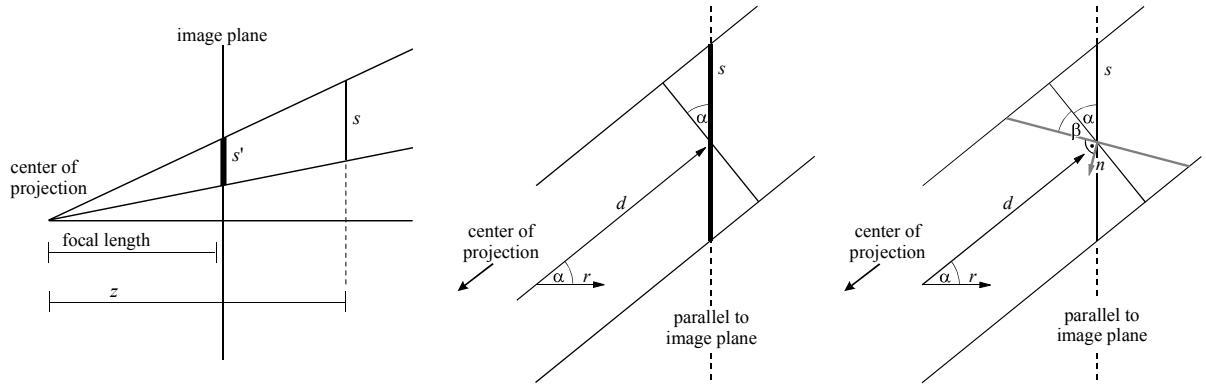
As sampling of a triangle with large projected area turns out to be more expensive than rendering using the conventional z-buffer algorithm, we identify triangles with large projected area in our group selection scheme and render these using the conventional algorithm (4), only small triangles will be replaced by sample points. This fallback strategy guarantees that the rendering time cannot drop below that of the conventional z-buffer for small scenes.

Image reconstruction: To reconstruct an image, first the occluded sample points have to be removed from the sample set. Afterwards, an image has to be constructed out of the visible points. 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 receives a sample from a foreground area, the foreground will overwrite occluded pixels. We investigate two further techniques in order to obtain higher efficiency (splatting) or higher image quality (filtering), respectively. Using *splatting* we reduce the number of sample points which must be processed by drawing larger point splats instead of pixels into the z-buffer, leading to a much higher performance at the expense of a reduced image quality. This enables near realtime rendering on conventional PC-hardware. With *filtering,* we improve the image quality by using a low-pass filter kernel to weight point sample contributions. This eliminates aliasing artifacts but leads to longer, non-realtime reconstruction times.

Dynamic scene modification: We have developed a static and a dynamic version of the walkthrough system. In the dynamic version, triangles can be inserted and deleted dynamically. The sole part of the data structure affected by the need for an efficient dynamic update is the distribution list used for sample selection. These can easily be replaced by dynamic data structures using balanced search trees.

3.2 Paper Outline

In the next chapter (section 4), we derive the probability distribution used as sampling density. Then we describe and analyze the reconstruction techniques (section 5), followed by the description of the approximation algorithm for sample extraction (section 6). The image reconstruction is described first because it sets the preliminaries for the point selection scheme. Section 7 describes two important extensions to the basic sampling algorithm. The first is an algorithm which avoids the sampling of large triangles by using conventional z-buffer rendering instead. The conventional algorithm is faster by up to two or three orders of magnitude for large triangles. The second extension is a hierarchical instantiation scheme. It provides a memory consumption linear in the size of a scene graph given as input to the rendering algorithm. Without this modification, it would not have been possible to encode scenes of sufficient complexity. Section 8 shows rendering results obtained from our prototype implementation and measurements of rendering times in dependence on the accuracy parameters of the approximate sample point selection algorithm. We conclude and describe possible future directions in section 9.



(a) segments parallel to the image plane **(b) segments orthogonal to the view direction** **(c) general segments**

Figure 2: perspective projection, figure (b) and (c) show magnified views of a very small surface element *s*, i.e. the local projection is nearly parallel.

4 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 in the image plane. Then we use this factor normalized by the integral over all objects in the viewing frustum as sampling density. A surface fragment s parallel to the image plane is scaled by $1/z^2$ where z is the depth of the object (figure 2a). If s is not parallel to the image plane, its orientation must be taken into account to determine the projected area: Objects perpendicular to the view vector d can be rotated to be parallel to the image plane. So they must be enlarged by a factor of $1/\cos\alpha$ so that they cover the same area on the screen as before (figure 2b) where α is the angle between the viewing direction r and the view vector d. This blow-up of the projected area towards the outer screen area is specific to the planar perspective projection (a spherical projection would not show such a behavior).

Surface fragments with arbitrary orientation have to be rotated to be perpendicular to the view vector (figure 2c), yielding a reduction of $\cos\beta$ of the projected area with β as the angle between the viewing direction and the surface normal. By convention, surfaces with an angle β of more than $\pi/2$ are regarded as invisible (backface culling). Thus the value of the cosine must be clamped to the interval [0,1], denoted here as applying the function $cut_{[0,1]}$ to it. Putting all this together yields a total factor of

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

by which infinitesimal small surface fragments are scaled by a perspective projection. To get absolute values, the projection factor must be multiplied by a constant to take into account the scaling of the image coordinate system and the focal length but we need relative values only here.

We now define the probability distribution by which the random sample points are to be chosen using the following density function:

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

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

5 Image Reconstruction

In this section, we assume that a set of sample points is given which were chosen according to the density function defined above (a strategy for choosing sample points will be developed in section 6). Using that sampling distribution will lead to a uniform distribution of sample points on the projections of the objects in the image plane and thus enables a reconstruction with fixed resolution. We will describe and analyze the basic reconstruction technique in section 5.1. Section 5.2 and 5.3 extend this technique in order to gain a higher reconstruction speed or a better image quality, respectively.

5.1 Per-Pixel Reconstruction

The per-pixel reconstruction algorithm takes all sample points in arbitrary order, projects them and draws them into a z-buffer: The z-values are initialized to infinite depth and the framebuffer to the background color. Then the depth of each sample is compared to the depth stored in the z-buffer at its position. If the depth of the sample is smaller than the stored value, the depth is written into the depth buffer and the shading function is evaluated at the sample point and written into the framebuffer. Otherwise, the sample is ignored.

To analyze the correctness of this method, we must give a definition of a correct image first: We consider one pixel of the final image. To make the description easier, we assume without loss of generality that every pixel is fully covered by triangles, i.e. there is no "background" leaking through. The triangle fragments seen through the pixel fall in two categories: Visible fragments and hidden fragments. An image is considered a *correct image of the scene* if every pixel shows a color found on one of the visible triangle fragments within the pixel. This definition of correctness abstracts from the slightly different behavior of rendering algorithms such as raytracing or z-buffering in choosing sub-pixel details.

To guarantee a correct image reconstruction in the sense of this definition, every pixel must receive at least one sample point from a foreground area and this point must have the smallest depth value among all sample points, because then it will overwrite all other sample points producing a correct pixel color.

We now proceed in two steps. In the first step, we show how many sample points must be chosen to guarantee that every pixel receives at least one sample point from a foreground area (section 5.1.1). In the next step, we identify the conditions under which this will also be the point with the smallest depth value (section 5.1.2). As this fact cannot always be guaranteed,

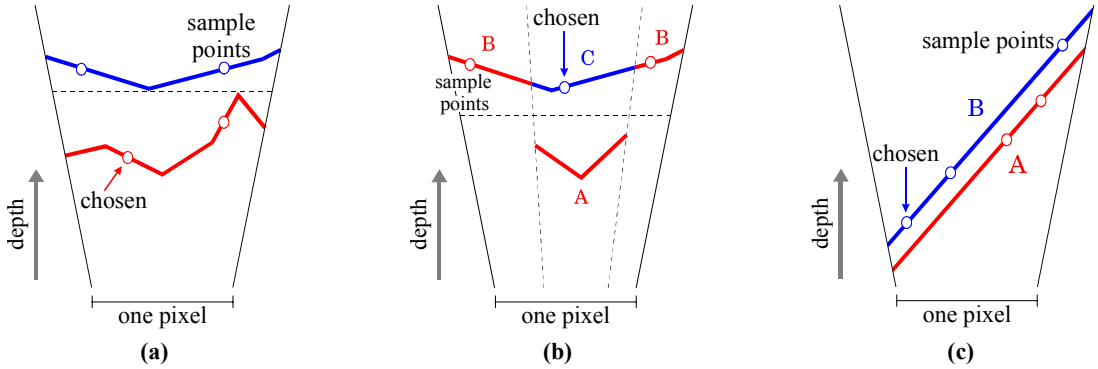


Figure 3: different occlusion situations

there will be some cases in which incorrect images will occur, but these can often be avoided easily in practice.

5.1.1 Sample Size

We now determine 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.

Every pixel is by definition fully covered by visible surface fragments. First, we assume that there are no hidden surfaces in the image. Then the probability that a visible sample is chosen from a pixel is the same for all pixels and the problem can be modeled as a simple urn model: Given v bins (with v denoting the resolution of the image in pixels), how many balls must be thrown randomly, independently, with equal probability into the bins until every bin has received one ball? This is an "occupancy problem" well known in the randomized algorithms literature. Motwani and Raghavan [23] show 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 further show that asymptotically, the probability for a deviation of $c \cdot v$ from the expected value drops off exponentially with c for large values of v. Thus, one can conclude that the size of the sample necessary to fill all pixels with high probability (say 99%) is $v \cdot \ln v + O(v)$ when no occlusion takes place.

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. The occluded area is now represented by h additional bins that are chosen with the same probability as the other v bins. It is now necessary to distribute $m_v = v \cdot \ln v + O(v)$ sample points among the v visible bins by throwing m_a balls randomly at all a bins. The probability that m_v balls go to the visible bins should be greater than a given probability s (say, again 99%). Let $p_v := v/a$ and $p_h := h/a$, then using the binomial distribution we obtain:

$$\sum_{k=m_v}^{m_a} \binom{m_a}{k} p_v^{\,k}\, p_h^{\,m_a-k} \geq s$$

To solve for m_a we approximate the binomial distribution by the Gaussian normal distribution and get:

$$\sum_{k=m_v}^{m_a} \binom{m_a}{k} {p_v}^k {p_h}^{m_a-k} \approx 1-\Phi\left(\frac{m_v - m_a p_v}{\sqrt{m_a p_v p_h}}\right) \geq s$$

where Φ is the Gaussian normal distribution function [12]:

$$\Phi(x) := \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{\tau^2}{2}} d\tau$$

The approximation is valid if m_a is large and neither p_v nor p_h are extremely small, which are reasonable assumptions here. Solving for m_a we obtain

$$m_a \geq \frac{1}{p_v}\left(m_v + \frac{\varphi^2 p_h}{2} + \varphi\sqrt{\frac{\varphi^2 {p_h}^2}{4} + p_h m_v}\right)$$

using φ to denote $\Phi^{-1}(1-s)$.

The function Φ quickly descends to zero for larger values of s (that means e.g. $\varphi \leq 4$ for $s=1-10^{-4}$) so it can be neglected in comparison with m_v. Using $p_h < 1$ we see that a value of

$$\frac{1}{p_v}\left(m_v + \varphi\sqrt{m_v}\right)$$

for m_a is sufficient to guarantee that m_v sample points are taken from the v visible bins. Substituting $v \cdot \ln v + O(v)$ for m_v we get:

$$\frac{1}{p_v}\left(v \ln v + O(v)\right) = a \ln v + O(a)$$

Thus a sample size of $a \cdot \ln v + O(a)$ is sufficient to guarantee that every pixel receives at least one sample from a visible object fragment.

5.1.2 Correctness

If every pixel receives at least one point sample from a visible triangle fragment, the correctness of the image cannot be guaranteed yet. We now distinguish two possible cases, only the first will always lead to a correctly reconstructed image:

1. "separated occlusion": The smallest interval covering the depth of all the visible fragments does not overlap with the smallest interval covering the depth values of all the hidden fragments (figure 3a). In this case, the point sample with the smallest depth value must stem from a visible triangle fragment and thus we obtain a correct reconstruction: It will overwrite all other points in the depth buffer.

2. "non-separated occlusion": If the depth intervals of visible and hidden fragments overlap, a false reconstruction of occlusion can occur because it is possible that the sample point with the smallest depth value belongs to an occluded triangle fragment. This point would incorrectly overwrite the sample points from visible areas within the pixel. 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 3b shows such an example: It is possible that pixels of region *C* occlude sample points from 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 severe losses of image quality have to be anticipated in this case because the surface attributes of the occluded area *C* often do not differ much from the neighboring regions. This situation leads only to "fuzzy" object boundaries.

The other situation is shown in figure 3c: Two parallel, extensive surfaces, which are not perpendicular to the viewing direction, lie in close proximity to each other. If the depth distance is very small, the hidden surface *B* might shine through with up to 50% probability. This phenomenon is often encountered when offset polygons are used to model surface details (figure 4). In this case, the problem can easily be avoided by converting these polygons into textures or removing the background underneath, as done for the city scene shown in section 0. Another possible situation where this effect can occur is when there are thin objects with front- and backfaces in the scene, e.g. the leaves in the tree shown in figure 5. These have dark backsides which can leak through if the relative distance to the camera is large enough. This problem can be avoided by applying backface culling, using the surface normals found in the sample points to discard sample points from backfacing surfaces (figure 6). As the backface-test can be performed quickly in object space (more quickly than a transformation into the image plane necessary to draw the sample point), it also leads to a higher rendering performance.

Using these heuristics, the flaws of the reconstruction process have no severe impact on the image quality in most practical applications. These problems are not limited to our approach, the correct reconstruction of occlusion is a problem of the point sampling approach in general.

5.2 Splatting

The per-pixel algorithm is too expensive to run at interactive frame rates on conventional hardware: To render e.g. an image of 640×480 pixels with an average depth complexity of two about 7,8 million sample points have to be processed. Especially, a perspective transformation must be performed for every point. Even on sophisticated graphics hardware, this task could take up to a few seconds.

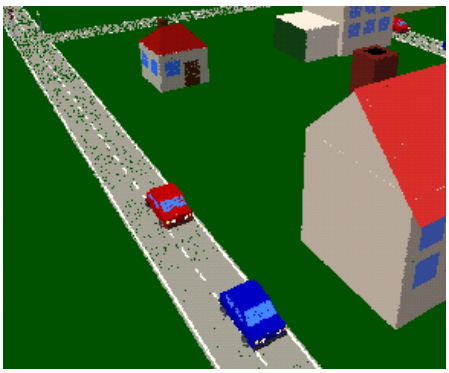


Figure 4: occlusion errors caused by offset polygons

Figure 5: the dark backsides of the leafs are leaking through

Figure 6: avoiding the effect using backface culling

A simple method for reducing the rendering costs is to use splatting: Instead of drawing single pixels, larger quadratic regions are filled with the same color and depth value, performing a depth compare on each pixel. To analyze the benefit of using colored squares of side length d we divide the image into a grid of cells with $\lfloor d/2 \rfloor$ side length. Then each square covers at least one such cell when it is drawn on the screen. Thus, the same arguments as in the section 5.1.1 can be applied to the cells instead of the pixels, which results in a sample size of

$$\frac{a}{(d/2)^2} \cdot \ln\left(\frac{v}{(d/2)^2}\right) + \mathrm{O}(a)$$

to be sufficient to cover all foreground regions. In practice, the size of the cell appears to have been chosen too pessimistically. Experiments show that one can safely substitute d for $d/2$ in the formulae above. As every square actually covers an area of d^2 and not $(d/2)^2$ on the screen this is also the reduction of the sample size one would have expected intuitively. Nevertheless, a formal proof seems difficult because the urn model from the preceding section can only be used as long as disjoint cells are considered because it requires independent events.

The image quality drops off with increasing d, but the results are always much better than those obtained by a simple reduction of the resolution by a factor d. We found a good compromise for values of $d \approx 2..5$. Using a value of $d = 2$ guarantees acceptable image quality even for critical scenes, saving about 75% computation time, while higher values may lead to objectionable visual artifacts. The effect depends on the scene to be rendered: Images with fine, high contrast details suffer more than images with smooth transitions between differently colored parts of the image (see figure 14 for example renderings).

5.3 Aliasing and Low-Pass Filtering

A reconstruction of a non-trivial image out of a finite sample leads to errors that arise in form of either structured aliasing artifacts or noise [14]. Cook [8] was the first to use random sampling to perform aliasing in the distributed raytracing algorithm. It shoots randomly jittered rays to trade off alias for noise that is usually less objectionable to the viewer than structured aliasing [14]. The amount of noise is then reduced by averaging over multiple random rays per pixel. Since our algorithm also uses random selection of surface points one could also expect converting some aliasing artifacts into noise, which can be eliminated by averaging over multiple frames.

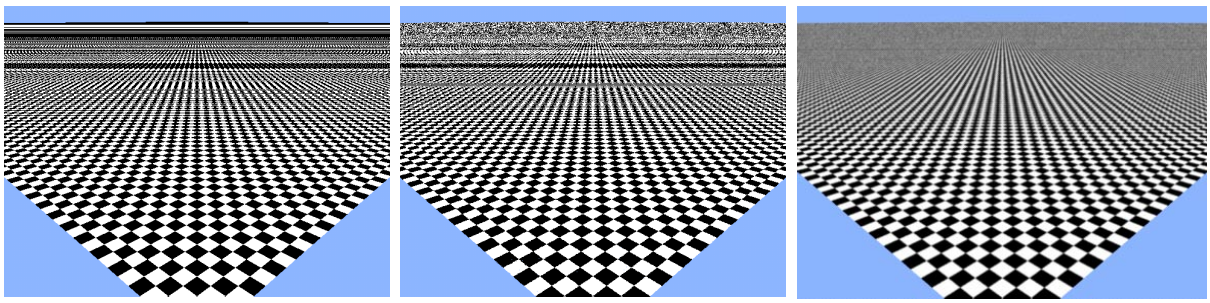
Figure 7b shows the rendering of a classical chessboard scene using per-pixel image reconstruction in contrast to conventional z-buffer rendering (figure 7a): The conventional z-buffer rendering shows severe aliasing artifacts for all polygons in the far field. The randomized algorithm shows similar artifacts only in the medium depth interval and unstructured noise for the polygons far away from the viewpoint. The aliasing artifacts in the near field are caused by the high oversampling: As every pixel receives $\ln v$ sample points on the average, in cases where there are only a few polygons per pixel there is a situation similar to conventional z-buffer rendering: Most pixels receive sample points from all polygons involved and the foremost is chosen to color the pixel. As the number of polygons increases, the choice of an $\ln v$ sized sample out of them becomes increasingly random and the aliasing disappears for unstructured noise (which can be removed by averaging).

To produce images with low aliasing artifacts, the reconstruction process has to be modified: After choosing the sample points we remove all points surrounding another sample point with lower depth value in a square of diameter *d*, similar to the splatting reconstruction method, but without discretization to pixel. The sample points are stored at full precision position and depth information in a two-dimensional array of buckets of size *d*. The adjacent 2x2 buckets of the bucket of a sample point are examined for fast retrieval of the surrounding points. To avoid the effect of systematically choosing the front-most point sample, we use a depth interval around each point of minimum depth in which no other points are discarded, assuming that these points all lie on the same surface. Then we calculate the color for each pixel as the weighted average of nearby pixels using a Gaussian filter kernel as weighting function, similar to [8].

This reconstruction method results in high-quality images with arbitrary small aliasing (figure 7c) but as the low pass filtering emphasizes low frequencies [14], a large sample size is necessary to avoid objectionable coarse granular noise artifacts. This leads to rendering times that are definitively outside the range of realtime applications (typically several minutes). Another drawback of this method is the depth interval heuristic used to avoid systematically choosing the foremost pixel: This value has to be estimated manually depending on the scene, wrong values may lead to a false reconstruction of occlusion. Backface culling is necessary to prevent misinterpretations on object boundaries.

6 Choosing sample points efficiently

The goal of this section is to develop an algorithm which chooses sample points randomly according to the probability density defined in section 4. The key observation is that an exact solution is not necessary. None of the image reconstruction techniques of the preceding section will fail if parts of the image contain *too many* sample points; they just cause additional processing costs. It must only be ensured that the sample density does not fall below the original sampling density anywhere. This leads to a conservative approximation algorithm which tries to come close to the optimal sampling density without ever falling below it. The running time of the reconstruction algorithm will then be increased proportional to the overestimation of the projected area. We will now present such an algorithm and prove that the additional costs are small under reasonable assumptions on the scene geometry.



(a) z-buffer **(b) randomized z-buffer** **(c) Gaussian filtering**

Figure 7: aliasing artifacts for different rendering / reconstruction methods

6.1 Distribution Lists

In a first step, we assume that all projected area values of the triangles are known in advance (they could be computed for each viewpoint using linear time). To choose random points according to that distribution we use *distribution lists*: The triangles are taken in arbitrary order and inserted into a list storing a pointer to the triangle and an additional summed-area value per triangle. We start with the projected area of the first triangle and increase this value with the projected area of each triangle inserted to obtain the discrete distribution function for the probability for choosing the triangles. The function is scaled by the total amount of projected area which is found as entry in the summed-area field of the last node.

To choose a random triangle we now determine a random number chosen uniformly from the interval [0,1] using a standard pseudo random number generator, scale the value by the total projected area and then search the first entry in the list whose summed projected area field is above the random value. Then we choose a random point from the triangle using a random linear combination of the vertices. This algorithm chooses an object according to the inverse of the distribution function defined by the summed projected area values of the triangles. It is a well known result from statistics that an 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 [25].

Using this technique, we can choose sample points in O(log n) time for n triangles if the projected area values are known. To avoid the costly computation we substitute estimated values for the exact values in the next section.

6.2 Approximation

Roughly speaking, our algorithm works as follows: The scene is divided into groups of triangles showing a similar projection factor (see section 4). The groups are obtained by dynamically choosing from a pre-computed hierarchy of triangle groups. Within each group, distribution lists are stored according to the absolute area of the triangles. For each group it is then determined how many points it must receive using the product of absolute area contained in the group and the upper bound known for the projection factor to estimate the projected area. Then, within each group, sample points are chosen using the distribution lists for the absolute area values. This method guarantees that the sampling density does not fall below the minimum value necessary.

To divide the scene into groups, three factors have to be bound since the total projection factor is a product of three independent terms: The *distance 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 latter distortion on the projection factor is relatively small, for typical values of the focal length parameter it can safely be ignored[1]. It remains to bound the two other factors.

[1] The deviation due to this distortion is also bound by the spatial subdivision used for bounding the distance factor so that even short focal length parameters do not lead to problems.

6.2.1 Spatial Classification

To bound the distance factor we build an octree for the triangles of the scene in a precomputation step. We can use "shortcuts" [3] to constrain the tree to storage requirements linear in the number of triangles, i.e. all subdivisions which do not divide triangles from each other are not stored explicitly. To handle triangles that intersect with the octree grid, we permit triangles to be stored at inner nodes if they lie outside the octree box with a distance of up to a constant proportion of the side length of the box.

For each node in the octree, a distribution list is provided for the absolute area values of all triangles contained in that node. For leave nodes the lists are computed from the triangles, inner nodes obtain their list as concatenation of the children's lists and the list for their own triangles in arbitrary but fixed order. Overall, 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. Therefore, the overall storage requirement is only $O(n)$ with n as the number of triangles and the precomputation can be performed in $O(n \cdot \ln n)$ time.

Every time the observer moves to a new position, a set of octree boxes is chosen out of the precomputed hierarchy for which the ratio between the minimum and the maximum projection factor is not larger than a given constant ε. Boxes that do not intersect with the viewing frustum are dropped. We use a recursive algorithm which traverses the octree from the root downwards and stops at each node which is small enough or lies outside the viewing frustum. Let τ be the ratio between the smallest and the largest depth value. We call this value the *relative depth range* of the scene. It is always bound by the ratio of the distance of the front clipping plane of the viewing frustum to the diameter of the scene. We will now show that the aforementioned algorithm only chooses $O(\log \tau)$ boxes from the octree using $O(\log \tau)$ time and that the cross-sectional area of the frustum of boxes selected exceeds that of the viewing frustum only by a constant:

Let z_{min} be the minimum value within an octree box and z_{max} the maximum value. Then the depth factor does not vary by more than ε if $z_{max}/z_{min} \leq \sqrt{\varepsilon}$. Let z_0 be the distance of the front clipping plane of the viewing frustum. We then divide the viewing frustum into $\log_{\sqrt{\varepsilon}} \tau$ depth intervals $[z_0\sqrt{\varepsilon}^{k}, z_0\sqrt{\varepsilon}^{k+1}]$ (see figure 8). Each section of the viewing frustum can be covered by octree boxes of diagonal diameter no smaller than the depth of the preceding section, because boxes of that diameter can only reach into the preceding section and so they do not vio-

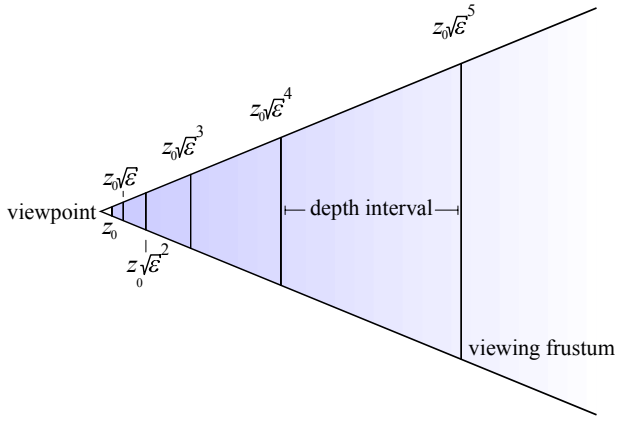


Figure 8: depth intervals

late the restriction of the maximum depth ratio.

Thus, the minimum size of the boxes is in a constant ratio to the depth of the current section. The volumes of the sections and the boxes up to a constant value both grow with the third power of the section depth, so a constant amount of boxes suffice to cover each section and thus only $O(\log \tau)$ boxes are needed altogether. As the diameter of the boxes is in a constant proportion to the sections depth value, the cross-sectional area of the viewing frustum is enlarged only by a constant due to the approximate clipping using octree boxes.

To bound the total number of boxes processed by the selection algorithm consider a box with diameter d which is further subdivided by the selection algorithm: To be neither selected nor culled this box must intersect with the viewing frustum in one of the sections demanding for a smaller box diameter. Now consider all sections with a depth interval below the diameter of the box considered. The volume of all these sections together differs only by a constant factor from the volume of the largest section because the sections sizes grow exponentially and thus the summed volumes make up a geometric progression. Thus, this area can also be covered with a constant number of boxes with size d. As the algorithm only processes boxes of $O(\log \tau)$ different sizes and only subdivides a constant number for each size the total number of boxes processed and therefore the running time is bound by $O(\log \tau)$.

The boxes selection algorithm has similarities to the methods proposed Rusinkiewicz and Levoy [28] and by Chamberlain et al. [6, 7]. Some ideas of the prove are taken from [6].

6.2.2 Expected Deviation

Using the spatial hierarchy only to assemble groups of objects with similar projection factor already leads to a satisfactory 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 equally distributed on the unit sphere. A worst-case analysis would not make much sense here: A scene filled with triangles perpendicular to the viewer could lead to an arbitrarily high overestimation in the worst case but such cases are rarely encountered in practice. We will now show that the expected value for the overestimation of the projected area is 4ε where $\varepsilon > 1$ is the parameter chosen in the box selection algorithm:

We fix the normalized view vector on the unit sphere and consider all vectors forming an angle of less than φ with it. The probability of a normal vector to lie within that region is given by

$$\text{Prob}(\varphi \le \varphi_0) = \frac{1 - \cos\varphi}{2}$$

as seen easily by elementary geometry. We obtain the probability density function f by differentiation as:

$$f(\varphi) = \frac{\sin\varphi}{2}$$

We then integrate over the exact orientation factor $\cos\beta$ weighted by the probability density function f to obtain the expected value of the projected area:

$$E(orientfact) = \int_0^{\pi/2} \cos\varphi \cdot \frac{\sin\varphi}{2} d\varphi = \frac{1}{4}$$

The expected value of the estimation is given by

$$E(est) = \int_0^{\pi} 1 \cdot \frac{\sin\varphi}{2} d\varphi = 1$$

so the ratio is 4. Multiplying with the factor ε accounting for the approximation of the distance factor leads to the value 4ε as claimed above. Note that the integration borders of the exact case take into account backface culling. As we can assume that sample points with backfacing normals can be removed from the sample with little costs (a vector difference and a scalar product) the expected size of the sample to be processed further is actually only larger by a factor of 2ε than the size of an ideal sample set.

6.2.3 Classification by Orientation

To archive a better estimation of the projection factor especially for scenes where the distribution of the surface normals is not uniform we use a simple classification scheme: We divide the unit sphere into classes of similar orientation by using a regular grid in polar coordinates[2]. Then for each class of orientation a spatial data structure is built as described in the preceding section.

To choose the spacing of the grid one must take into account the deviation of the orientation due to the position in space: The angle β in the orientation factor is defined as the angle between the view vector, i.e. the vector from the surface point to the viewer, and the surface normal. Therefore, the angle changes when the surface point is moved perpendicular to the viewing direction. Thus, it is necessary to restrict the spatial location of the objects to enhance the accuracy of the estimation of the projection factor. The possible angular deviation is equal to the angle under which the octree boxes extends are seen from viewpoint. This value can be bound by the ratio of the diameter of the box to its minimum depth distance to the viewer. This value is implicitly bound by the spatial subdivision algorithm because it bounds the ratio of the minimum to the maximum depth of the box and the boxes are symmetric. Let now λ be the maximum angle between two points in an octree box and the view point and we choose the grid spacing of the angle classes to be λ, too. Then the overall angular error is bound by 3λ and the expected overestimation of the orientation factor can be bound by:

$$E(est)/E(ideal) = \int_0^{\pi} \frac{1}{2}\sin(\varphi) \cdot cut_{[0,1]}\left(\cos(cut_{[0,1]}(\varphi - 3\lambda)\right) d\varphi \Big/ \frac{1}{4}$$

$$= \int_0^{3\lambda} 2\sin(\varphi) d\varphi + \int_{3\lambda}^{\pi/2+3\lambda} 2\sin(\varphi)\cos(\varphi - 3\lambda) d\varphi$$

[2] This is not optimal, a better solution could e.g. use a subdivision of a platonic primitive to obtain a more uniform cell size.

$$= 2 - \cos 3\lambda + \frac{\pi}{2} \sin 3\lambda$$

An analysis of this function shows that the expected overestimation decreases roughly linearly with λ from 2 to 1 when λ is decreased to zero while the costs for selecting boxes increase with λ^{-5}: When λ is reduced, the number of orientation classes grows with λ^{-2}. For small angles, the relative size of an octree box, i.e. the ratio of its size and depth, and the angle under which it is seen from the view point are roughly proportional to each other. So a reduction of the viewing angle λ leads to an increase of the number of octree boxes of λ^{-3}. The strong growth of the number of boxes for small values of λ restricts the number of angle classes which can be used: Only a rough angular subdivision is possible without strong increases of the running time of the box selection algorithm.

6.2.4 Worst-Case Scenes

The approximation algorithm used here to choose the sample points can lead to an arbitrary high running time in the worst case. Indeed any approximation based on regular spatial subdivision cannot handle worst-case scenarios with bounded costs:

Consider the scene shown in figure 9: It consists of a pile of several triangles perpendicular to their view vectors and a nearby triangle with the same orientation which is not perpendicular to the viewing direction. As long as theses triangles are kept in one spatial group, the overestimation of the projected area can be arbitrary large because there can be arbitrary many triangles in the pile. To divide the triangles seen from different angles from each other the relative size of the spatial subdivision boxes must be made smaller than the angle γ between the two elements seen from the view point. As this angle can be arbitrary small, this can also lead to arbitrary high costs. The conclusion is that regular hierarchical subdivision cannot handle worst-case scenes at bounded error and running time.

In practical applications such scenes are rarely found, but beside scenes with a relatively uniform distribution of surface orientations there are also scenes which show some preferred orientations: For example a scene showing a plane raising up to the horizon clearly has a fixed projected area. If the orientation to the viewer is not taken into account, we get a projected area of

$$\Theta\left(\int_{\Theta(1)}^{r} z \frac{1}{z^2} dz \right) = \Theta(\log r)$$

with r as the diameter of a circular plane segment, which is unbounded. But as the size only

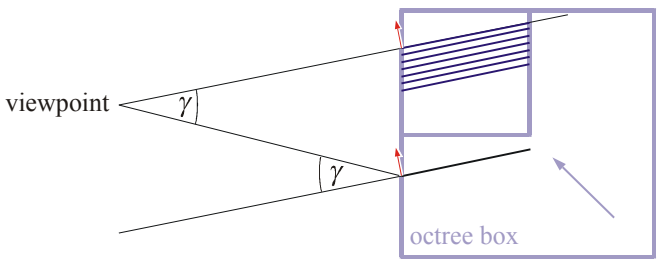


Figure 9: worst-case scene

grows logarithmically with the diameter, the overestimation is not very severe as long as the segment is not extremely large.

This argument can also be applied to similar cases: Many scenes showing urban landscapes contain groups of objects lying on such planes like walls or the ground plane. A relatively small number of angle classes are sufficient to separate the main orientations (roof, walls, ground) from each other. Then only a rough spatial subdivision is necessary to bound the overestimation of the projected area caused by the variation of the viewing angle due to a variation of the location of the objects because the overestimation only grows slowly with the spatial size of the object groups.

6.3 Dynamic Updates

A dynamic version of the data structures for sample extraction can be obtained applying some minor modifications to the static data structures. In the following, we will give a dynamic version of the spatial subdivision data structure. The generalization to orientation classes is straightforward.

The main problem is that the distribution list storing the accumulated area values in the static case cannot be updated efficiently. Therefore, we substitute a dynamically balanced search tree for the distribution list, e.g. a binary AVL-tree [22]: The child nodes of the tree store the area values of the triangles and the inner nodes store the sum of their child nodes. The entries in the tree are ordered by an inorder traversal of the octree. Therefore, each octree node can mark up its region in the tree by two pointers to leaf nodes in the search tree. The search algorithm which selects a sample point now at first runs upwards in the tree from the leaf nodes, summing up all area to the left and to the right of the search interval. A random number from the range [0,1] is taken and it is scaled by the total area in the tree minus the area outside the search interval. Then the search algorithm starts from the root searching the leaf node: At each inner node the first child is chosen, if its area value is not smaller than the search value. Otherwise, the second child is chosen and the search value is decremented by the area value of the first child.

When a dynamic insertion or deletion takes place, at first the octree must be updated, which can be done in $O(h)$ time, if h is the height of the tree. Then a leaf node is added or removed from the AVL-tree. The summed area values of the parent nodes must be corrected, taking $O(\log n)$ time and the tree must be rebalanced. The balancing operations of an AVL-tree preserve the order of the tree, so the intervals of the octree nodes are not affected by this step. The summed area values must be corrected for all nodes which are rotated during balancing, but this can be done locally in $O(1)$ time per node. $O(\log n)$ balancing operations are necessary, so the overall update time[3] is $O(\log n + h)$, the sample selection time remains $O(\log n)$ as in the static case.

The implementation of a separated balance search tree can be avoided if the scene consists of objects which are uniformly distributed in at least one dimension. Then the octree itself is relatively well balanced and it can be used as distribution tree: In each leaf node, the area val-

[3] If the octree node in which the object is to be inserted or from which it is to be deleted is known in advance, the $O(h)$ search for the node of the object is not necessary and the octree can be updated in $O(1)$ time: Due to the use of shortcuts, only local modifications are necessary. This results in a total dynamic update time of $O(\log n)$.

ues of the triangles are stored. In each inner node, a list with the eight summed area values for the children is 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 current search value. The summed area value of the child node directly before that node is subtracted from the current search value. So the correct leaf node can be found in $O(h)$ time. Dynamic updates can be done in $O(h)$ time, too: After adding or deleting a node from the octree in $O(h)$ time an additional traversal from the leaf affected to the root node is necessary to correct the summed area values.

For practical applications, the second variant is probably the more interesting one, because although the first variant is asymptotically faster, it 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 $O(h)$ may be acceptable. The experimental results in section 6.2.4 confirm a good behavior in practice: For scenes with uniformly distributed objects, sample selection using the octree as distribution tree is nearly as fast as in the static case.

6.4 Analytical Results

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

Let S be a three dimensional scene with surface normals uniformly distributed on the unit sphere, a the projected area of the scene, v the resolution of the rendered image, τ the ratio between the diameter of the scene and the minimal viewing distance and n the number of triangles in the scene. Then the scene can be rendered in $O((a \cdot \log v + \log \tau) \cdot \log n)$ time using $O(n)$ space for precomputed data structures. In all pixels of the image which are fully covered by foreground objects and in which the depth interval of foreground and background objects are disjoint a correct image is obtained with a high probability given arbitrarily.

Specific running time for plain landscape scenes: An important special case are scenes 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 landscape scenes. The projected area of such a scene can be estimated as follows:

$$A_{prj} \cong \int_{\Theta(1)}^{r} \Theta(z) \frac{1}{z^2} dz = \int_{\Theta(1)}^{r} \Theta\left(\frac{1}{z}\right) dz = \Theta(\log r)$$

The number of objects grows with r^2, so if these scenes are rendered with the randomized z-buffer algorithm without any further occlusion culling we obtain a total running time of $O((\log r \cdot \log v + \log r) \cdot \log r^2) \subseteq O(\log^2 r)$. This is a big improvement in comparison to the conventional z-buffer algorithm, which would require $\Theta(r^2)$.

7 Extensions

7.1 Using Conventional z-Buffer Rendering for Large Triangles

For each pixel on the screen, $\ln v$ sample points have to be projected into the image plane, where v is the resolution of the image. Thus, for large 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 but it would be more efficient to use the conventional z-buffer algorithm. For large triangles, the conventional algorithm is faster by about three orders of magnitude because it can be implemented using fast incremental integer arithmetic to fill the pixels of the triangle. 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 estimation of the projection factor for the triangle and the absolute area of the triangle. The first factor is known exactly, it is determined by the octree box selection algorithm. To estimate the second factor we use classes of triangles with a similar area. Each class stores area values which differ only by a constant factor (say two or four) from each other, so 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 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 shares its depth- and framebuffer with the reconstruction algorithm of the point sample renderer. Setting a conservative value for the threshold value, it can be ensured that the rendering speed never drops below that of a conventional z-buffer renderer.

The drawback of this approach is that additional classes for triangles of different area have to be used. Together with orientation classes and the number of boxes selected by the spatial subdivision algorithm a considerable overhead for box processing could emerge, as all three factors multiply. But this is no severe problem in practice: Experiments show that in most cases the use of classification by orientation does not lead to significant performance improvements for the sampling algorithm (see section 8.2). So one can use classification by area *instead* of classification by orientation and in this constellation the number of boxes which must be processed is small enough.

7.2 Scene Encoding

The running time of the randomized z-buffer algorithm shows a very weak dependence on the number of triangles in the scene. Thus, the scene complexity which can be handled lies some orders of magnitude higher than typical input scenes for the conventional algorithms. Therefore, the main barrier for the display of extremely detailed scenes is the memory consumption of the scene description. Using an explicit storage of all triangles does not even lead to a

scene complexity at which the pure sampling algorithm is significantly faster than the conventional z-buffer algorithm because of the large sample size required (see section 5.1.1).

To be able to encode meaningful scenes of high complexity, we have adopted a scene graph based hierarchical instantiation scheme[4]: The scene is described by a conventional scene graph as found in many software libraries [33]. Reference nodes are used to define multiple instances of the object described by a subgraph. The instances can differ from the original by a combination of scaling, rotation, reflection and translation[5]. Subgraphs starting at nodes which have multiple incoming references can be marked by the modeler to be regarded as "*subobjects*". For each such subobject, a private spatial subdivision structure is constructed. Instead of all the triangles contained in the subobject, only a dummy node is inserted into the spatial subdivision structure of the parent object. The dummy node contains a pointer to the private octree and a transformation matrix. This scheme is carried on hierarchically allowing sub-subobjects within subobjects. The resulting data structure is used like a conventional octree for the whole scene: When the box selection algorithm steps over a dummy node, the current camera settings are transformed by the inverse transformation of the node and the algorithm is continued in the substructure. When the point selection algorithm reaches a substructure instead of a triangle it is continued in the sub distribution list and the transformation found in the dummy node is applied to the sample point. As every instantiation layer leads to one additional matrix/vector multiplication, 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).

We consider this solution to the scene-encoding problem only a first step. To be able to encode large scenes for the rapid display by output-sensitive rendering algorithms further techniques have to be developed. We believe that the scene encoding problem will become increasingly important in the near future with the advent of powerful rendering hardware and output-sensitive rendering algorithms.

8 Implementation and Results

We implemented a prototype of a walkthrough system using our algorithm to examine image quality and rendering performance in practice. The system was implemented in C++ using OpenGL for low-level rendering and tested on an 800Mhz Athlon processor system with an nVidia GeForce-2 GTS graphics board. The z-buffer renderer used for comparison and for combination with point sample rendering was not optimized, especially, it did not make use of triangle strips. However, as the point sample renderer also leaves some opportunities for optimization, we think that the comparison remains fair.

[4] We will describe the application to the spatial subdivision structure only. Once more, the generalization to orientation classes and area classes is straightforward.

[5] We restrict the transformation to these operations because they can be mapped to manipulations of the camera settings. This is used by the traversal algorithm.

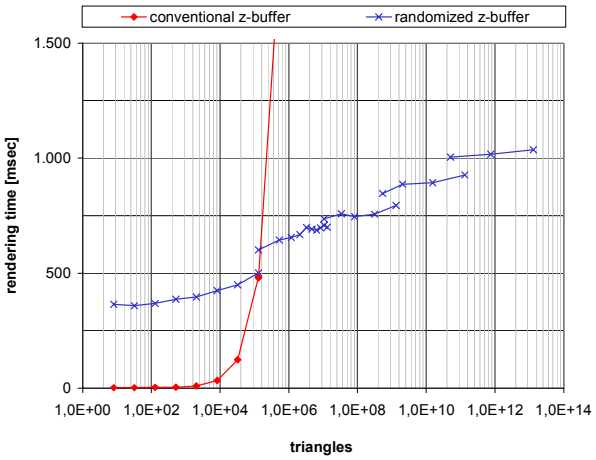


Figure 10: running time for different scene complexities

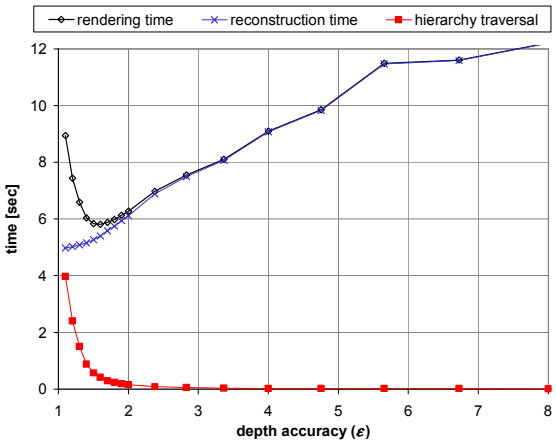


Figure 11: trade-off between traversal time and reconstruction time

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. This constellation ensured that the projected area and other geometric parameters remained constant for all levels of detail. The results can also be applied to scenes with a complex geometry like fractally-modeled trees or landscapes but these have not been used as benchmark scenes in order to preserve the orthogonality of the results. The scene was rendered using point sample rendering only, the automatic invocation of conventional z-buffer rendering was disabled.

Figure 10 shows the running time for an exponentially increasing scene complexity. The randomized algorithm shows a roughly linear increase of running time approving the logarithmic growth of its running time with the number of triangles. The discontinuities show up at points where additional instantiation layers had to be introduced. Overall, the rendering time was only increased by about a factor of three between 32 triangles and 10^{13} triangles, which clearly shows the strong 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 the conventional z-buffer rendering. For a small number of triangles, the running time of the random sampling algorithm is higher than that of the conventional z-buffer algorithm because it has to compute a constant amount of perspective projections depending only on the projected area. This motivates to render large triangles using the conventional z-buffer algorithm as described in section 7.1. If this strategy is applied, the combined algorithm is always at least as fast as the conventional algorithm.

8.2 Sampling Parameter Optimization

The two main parameters of the sample distribution process are the depth accuracy ε and the granularity of the angular subdivision λ. We varied these parameters for different benchmark scenes to identify optimal settings. Classification by triangle area for automatic conventional z-buffer rendering of large triangles was deactivated here, too.

Depth accuracy: Choosing the parameter ε is a trade-off between box selection time and image reconstruction time. The number of boxes used grows rapidly with decreasing ε: The depth sections which constrain the side length of the boxes in the proof in section 6.2.1 start at depth values of ε^k with increasing k. Therefore, the relative depth, i.e. the ratio between the minimum and maximum depth of one section, is proportional to $(\sqrt{\varepsilon}\text{-}1)$ and thus the volume of one box decreases with $(\sqrt{\varepsilon}\text{-}1)^3$ leading to costs of $\Theta((\sqrt{\varepsilon}\text{-}1)^{-3})$ for the box selection algorithm. On the other hand, a large value for ε leads to a poor approximation of the ideal sample distribution and thus to increased image reconstruction costs: In the worst case the depth factor can be overestimated by a factor ε. In an average case, assuming a uniform distribution of the triangles over the scene the expected overestimation is just $\Theta(\sqrt{\varepsilon})$: Averaging over the over-estimation factor 1:1/z for a depth section reaching from a to $a\sqrt{\varepsilon}$ we get:

$$\frac{1}{a\sqrt{\varepsilon}-a}\int_a^{a\sqrt{\varepsilon}} z\,dz = \frac{a(\varepsilon-1)}{\sqrt{\varepsilon}-1} \in \Theta\left(\sqrt{\varepsilon}\right)$$

The sum of these two monotonic costs has an absolute minimum at a value ε which depends on the relative depth range τ of the scene, which scales the first cost function, and on the projected area, which scales the second cost function. Figure 11 shows measured running times for these components for an example scene which approve the theoretical analysis: The trade-off between the running times of the two steps of the randomized z-buffer algorithm can clearly be seen as well as the strong growth of the running time for small values of ε and the weaker growth for large values.

For practical applications, it would be desirable to determine the optimal value automatically. As there is only one local minimum, this could easily be accomplished by using a minimum search algorithm, like a gradient descent algorithm. As long as the optimal value is unknown, it is be advisable to use a relatively large value for ε because of the strong growth of the running time for small values of ε.

Angular subdivision: The optimal value for the grid spacing λ of the angle classes also heavily depends on the scene material to be rendered. We tested several scenes with quite uniformly distributed surface normals. For views of scenes which require only moderate spatial subdivision, the rendering time can be improved by angular subdivision. If many boxes are necessary to achieve a small overestimation of the depth factor (e.g. if the scene is seen from a view point with large relative depth range), additional angular subdivision even increases the running time. Improvements were only possible as long as the relative depth range of the scene was very small. Thus, for most applications, especially for walkthroughs of extended

replication	1×1	2×2	3×3	4×4	5×5	6×6
number of triangles	33,950	135,800	305,550	543,200	848,750	1,222,200
initial precomp. time [s]	1.87	8	19	34	52	84
insert 5804 triangles [s]	0.562	0.546	0.582	0.580	0.581	0.572
delete 5804 triangles [s]	0.011	0.010	0.010	0.012	0.025	0.017
throughput (static) [pts/s]	562,393	543,758	539,103	531,011	524,006	511,417
throughput (dynamic) [pts/s]	479,592	461,125	449,806	435,024	416,132	399,548
overhead	17%	18%	20%	22%	26%	28%

Table 1: dynamic updates of scenes of different sizes

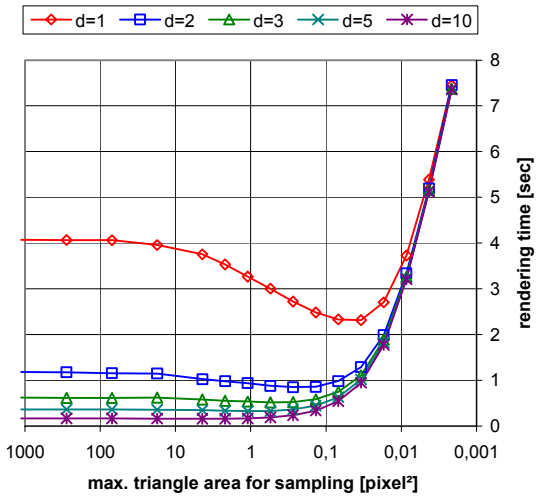


Figure 12: z-buffer rendering of large triangles, different threshold values

Figure 13: a cow model inserted into a city scene

virtual environments, classification by orientation should not be used.

The reason for this somehow surprising result is that using angular subdivision multiplies the number of boxes to be processed by an additional factor and thus limits the number of boxes that can be spent to bound the overestimation of the depth factor. As the average overestimation of the orientation factor is only a factor of two (uniformly distributed normals, taking backface culling into account), constraining the depth factor is more profitable than constraining the orientation factor.

8.3 Combination with Conventional z-Buffer Rendering

Two parameters are open in the triangle area classification scheme described in section 7.1: The size of the area classes and the threshold value for the estimated projected area at which conventional z-buffer rendering is invoked. In practical experiments, the value for the size of the area classes had little impact on the running time. Values between 2 and 64 lead to similar rendering times when all other parameters where chosen optimally. The reason for this is that firstly, the accuracy of the estimation of the projected area has little impact on the running time, as it affects only a small "border" region of the scene. Secondly, the size of area classes does not affect the total number of boxes to be processed as severely as the size of the angular classes because only a one-dimensional domain is subdivided.

For our test scenes, the optimal value for the threshold value at which the algorithm switches to conventional z-buffer rendering lay, as theoretically expected, in the range of the pixel size divided by the oversampling ratio $\ln v/d$ and it was roughly independent of the scene material and camera position. Figure 12 shows the rendering time in dependence on the threshold value for the city scene shown in figure 14f. The minimum of the rendering time varies with the splat size d used for image reconstruction. The gain achieved by the integration of z-buffer rendering naturally depends on the sampling cost. This leads to a most pronounced minimum for the per pixel reconstruction ($d = 1$) and to smaller gains for larger splat sizes.

8.4 Dynamic Updates

We implemented the simplified variant of the dynamic distribution tree described in section 6.2.4, i.e. the variant which stores additional summed area values in the spatial octree and uses this structure to choose sample points. For practical implementations, this variant is more appealing than the "exact" implementation using dynamically balanced search trees because it causes only a small implementation overhead in comparison with the static version. We used the city scene from section 8.5 replicated on a quadratic grid (without using instantiation) obtaining a maximum object count of 1.2 million triangles. Due to the large memory consumptions (175MB in the static and 363MB in the dynamic case), this scene complexity can be considered the upper limit of objects which can be managed without using instantiations. If instantiations are used, the dynamic operations just have to be applied to the sub-search structure affected, which then also falls below this maximum size.

We inserted a cow model (taken from [13]) consisting of 5804 triangles into the scene (see figure 13) and removed it again. The resulting running times are shown in table 1: Rebuilding the data structure from the scratch takes up to 84 seconds while the dynamic update only takes half a second for insertions (100µsec / obj) and up to 25 milliseconds for deletions (4,3µsec / obj)[6]. The update times are short enough to allow for interactive local modifications of small parts of the scene typical for interactive editing applications. If instantiations are used, only one deletion and insertion operation is necessary to change the transformation of a complex object with a private distribution tree (the dynamic data structure does not distinguish between subobject entries and triangle entries). As only one entry in the parent distribution tree has to be updated, this common operation can be performed within less than a millisecond.

The critical point of the simplified implementation of the dynamic distribution tree is the performance of the data structure for choosing sample points. We compared the sample point throughput per second of the dynamic version with that of the static version. As the static version uses an optimally balanced binary search in a static distribution list in contrast to a search in the unbalanced octree, a higher performance could be anticipated. The last row in table 1 shows the percentage by which the octree distribution version is slower than the static version. The overhead grows with increasing size of the scene, but it does not exceed 30% for the largest city model. We tested some other scenes with roughly uniformly distributed objects[7] and similar object count and also observed a performance advantage of no more than 30% for the static version. Thus, the simplified version of the dynamic algorithm can be recommended for practical applications, as the losses of rendering speed are not severe as long as the objects are roughly uniformly distributed.

[6] Insertions are slower than deletions because our implementation limits the maximum number of objects stored in one node to 64 but the minimum is only limited by one. This leads to more node split operations during insertions.
[7] We did not test adverse scenes as it can be seen analytically that the complexity will be linear in the worst case.

Scene	complexity [triangles]	memory consumption	precomputation time	rendering time $d=1$	$d=2$	$d=5$	$d=10$	z-buffer
"Happy Buddha"	1,085,634	163 MB	129 sec	1.54 sec	0.44 sec	0.096 sec	0.034 sec	1.18 sec
city scene	3,395,000	7.5 MB	3 sec	1.7 sec	0.57 sec	0.23 sec	0.12 sec	4.33 sec
landscape scene (diffuse)	410,501,184	137 MB	123 sec	6.1 sec	1.8 sec	0.64 sec	0.26 sec	7 min 51 sec
landscape scene (phong)	410,501,184	166 MB	123 sec	19.2 sec	6.9 sec	4.5 sec	4.0 sec	8 min 59 sec
replicated trees (close-up)	$1.37 \cdot 10^{14}$	36 MB	21 sec	6.1 sec	1.7 sec	0.58 sec	0.23 sec	> 160 d (est.)
replicated trees (overview)	$1.37 \cdot 10^{14}$	36 MB	21 sec	9.1 sec	4.6 sec	1.3 sec	0.44 sec	> 160 d (est.)

Table 2: rendering times for different scenes and different splat sizes d ($d=1$ is equivalent to per-pixel reconstruction)

8.5 Example Renderings

We applied our algorithm to three different example scenes to demonstrate the benefits of our algorithm. All images were rendered in a resolution of 640×480 pixel (table 2 summarizes the results):

Replicated trees: A scene with $5.9 \cdot 10^9$ instances of complex tree models replicated on a quadratic grid was rendered to show the ability to handle scenes of extreme complexity. The scene consisted of $1.4 \cdot 10^{14}$ triangles. A conventional z-buffer rendering would have taken about 160 days (assuming an optimistic triangle throughput of 10^7 triangles per second). The randomized algorithm delivers good results after a few seconds. Figure 14a shows a close-up of the scene. It took 6.1 seconds to render using per-pixel reconstruction. Figure 14b shows an overview over the scene. It was rendered 10 times with per-pixel reconstruction in 9.4 seconds each; the image shown is the arithmetical per-pixel average of all ten frames. It can be seen well that the random point sample rendering leads to averaged color values in the far field giving a natural visual impression of the over-detailed geometry.

Landscape scene: We placed instances of the tree models used in the tree scene on a fractal landscape model to show that visually appealing models can be rendered using our technique. The complete landscape model consisted of about 410 million triangles. A high quality rendering (figure 14c) took 19 seconds while a conventional z-buffer rendering took about 9 minutes, showing a significant speedup without visible differences between the two rendering results. Near realtime rendering performance could be achieved at reduced image quality: We substituted a static diffuse illumination model (as used in all other examples) for the relatively expensive view dependent phong illumination model and enlarged the splat size. This led to a rendering time of 640 msec for $d=5$ and 260 msec for $d=10$. Such renderings with degraded image quality (see figure 14d) may be used as interactive previews within a scene editing application. We used this technique for example to place the camera positions for our example renderings.

"Happy Buddha" mesh: In this scene, a highly detailed mesh of simple topology was rendered to show that the image quality of our algorithm is comparable to other approaches, such as multi-resolution modeling. Some algorithms which perform well on unstructured scenes like forest scenes have severe problems on smooth meshes [7]. Figure 14e shows the rendering results for the well-known "Happy Buddha" mesh (also taken from [13]) using the point sample rendering algorithm (automatic z-buffer rendering was disabled). The image quality at small splat size ($d=2$) is nearly indistinguishable from a conventional z-buffer rendering but already considerably faster. However, it should be noted that higher speedups can be achieved

using mesh simplification algorithms (see e.g. [13]) because the model still contains large amounts of redundant triangles.

City scene: A city scene consisting of a huge amount of simple objects was rendered with automatic z-buffer rendering for big triangles to demonstrate that near realtime performance can be achieved for such types of scenes. The city scene consists of 3,395,000 triangles. A conventional z-buffer rendering took 4.24 seconds. Using the point sampling algorithm for small triangles, we were able to achieve frame rates of 2 to 5 frames per second, depending on the image quality: 560 msec for a splat size of $d=2$, 225 msec for $d=4$. This was close to realtime performance while delivering an acceptable image quality (figure 14f). Many other approximate rendering approaches would have had difficulty in representing the large number of independent objects in the far field of this scene within a short time. For example, mesh simplification would not have been applicable because all individual objects were already very simple and further simplification would have led to unacceptable errors. We enlarged the scene model by replication on a 100×100 grid, so that the new scene contained about $3{,}4\cdot10^{10}$ triangles. This led to a rendering time of 963 msec for $d=4$ respectively 2,97 sec for $d=2$. So the rendering time increased only by a factor of about 4 while a conventional z-buffer rendering would have taken 10,000 times as long (i.e. 11h). This shows the impact of the logarithmic asymptotic behavior of our algorithm: As shown in section 6.4, the rendering time grows only with $O(\log^2 r)$ when r is the diameter of the scene, instead of $O(r^2)$. Thus, much larger scenes are manageable.

9 Summary and Future Directions

We presented a new output-sensitive rendering algorithm which is based on a new technique of choosing random sample points in object space which are approximately uniformly distributed in screen space. This technique leads to asymptotically efficient rendering under sensible assumptions on the scene geometry, it allows for efficient dynamic updates and delivers an image quality which is in most cases comparable to a conventional z-buffer rendering. Practical examples show that scenes of high complexity can be handled: The rendering speed and the dynamic update performance is high enough for interactive editing applications on conventional PC-hardware. In certain cases with low projected area to be sampled (like the city scene when using conventional z-buffering for the larger triangles) near realtime performance can be achieved facilitating interactive walkthrough applications.

Our approach leaves some open problems to be solved in future work:

Realtime rendering: The rendering performance could be strongly enhanced if the oversampling could be reduced. We are planning to use multiple sample caches for regions of different distance to the viewer which collect sample points and discard redundant ones. These caches could be used for several subsequent frames, so a higher rendering performance could be achieved for walkthrough situations.

Occlusion culling: We are also currently working on an algorithm which uses statistical information about how many sample points from one octree box have been accepted by the z-buffer test to decide to hierarchically cull occluded portions of the scene.

Modeling: To be able to handle complex scenes it is necessary to encode them in a data structure with reasonable storage requirements but with full access for the point sample renderer. Here, the next step could be to provide direct sampling of higher order primitives like parametric spline surfaces (instead of triangles) and of models defined by simple generative models, like the replication of geometry on parametric surfaces. Subdivision in parameter space for the classification by projection factor and parametric distortion could serve as basic technique to implement such modeling techniques.

High quality rendering (offline): To be able to use the randomized z-buffer algorithm in high quality rendering applications, some problems have to be solved, especially: efficient antialiasing with adaptive oversampling, depending on the local variation of intensity, a correct handling of transparency in partially occupied regions and output sensitive solutions to global illumination problems.

Acknowledgements

The authors wish to thank Prof. Straßer of Tübingen University for his support of this work. We also wish to thank Tamás Lukovszki, Ingmar Peter and Christian Sohler for valuable comments and discussions and Mario Vodisek for providing additional models for the city scenes.

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.,* pp. 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] Catmull, E.: *A Subdivision Algorithm for Computer Display of Curved Surfaces.*Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.

[6] Chamberlain, B., DeRose, T., Lischinski, D., Salesin, D., Snyder, J.: *Fast Rendering of Complex Environments Using a Spatial Hierarchy*. Technical Report, UW-CSE-95-05-02, University of Washington, 1995.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Debevec, P.E., Taylor, C.J., Malik, J.: Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In: *SIGGRAPH 96 Proceedings, Annual Conference Series, 11-20*, 1996.

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

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

http://graphics.cs.uiuc.edu/~garland/CMU/quadrics/quadrics.html

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

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

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

[17] Heckbert, P. S., Garland, M.: Survey of Polygonal Surface Simplification Algorithms. In: *SIGGRAPH 97 course notes,* 1997.

[18] Klein, R.: Multiresolution representations for surfaces meshes. *Computer & Graphics, 22(1)*, 1998.

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

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

[21] Maciel, P.W.C., Shirley, P.**:** Visual Navigation of Large Environments Using Textured Clusters. In: *Proceedings of the 1995 ACM SIGGRAPH Symposium on Interactive 3D Graphics, 95-102,* 1995.

[22] Manber, U.: *Introduction to Algorithms, A creative Approach.* Addison-Wesley 1989.

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

[24] 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.

[25] 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.

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

[27] 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.*

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

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

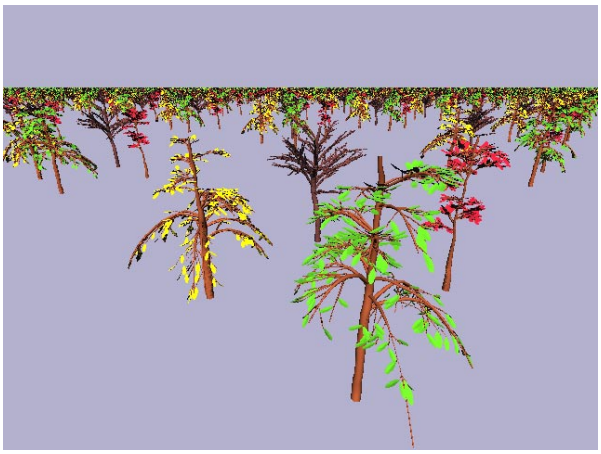
[30] Shade, J., Lischinski, D., Salesin, D. H., DeRose, T., Snyder, J.**:** Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In: *ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH 96 Proceedings),* 1996.

[31] 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.

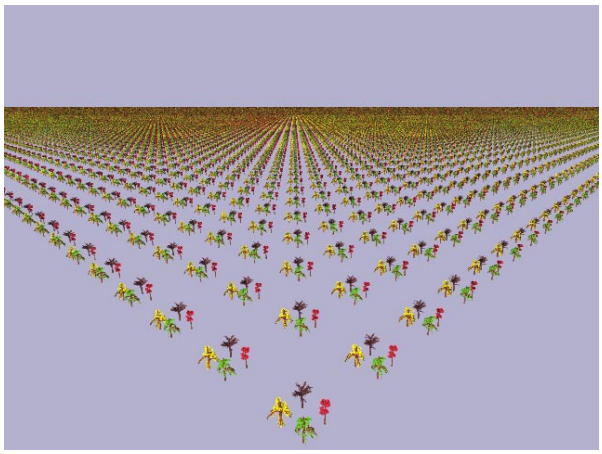
[32] Teller, S.J., Séquin, C.H.: Visibility Preprocessing For Interactive Walkthroughs. In: *Computer Graphics (SIGGRAPH 91 Proceedings), 25 (4)*, 61-69, 1991.

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

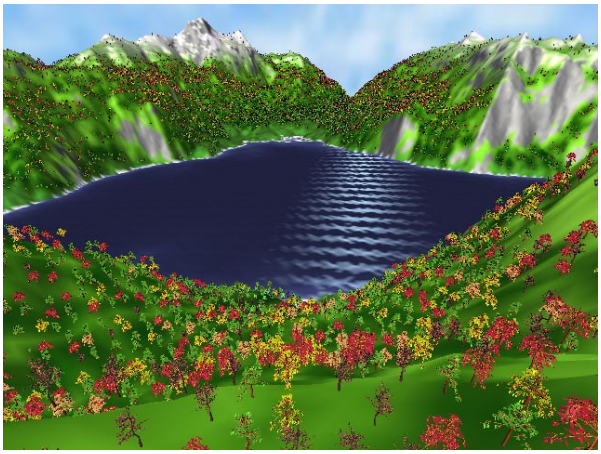
[34] Zhang, H., Manocha, D., Hudson, T., Hoff, K.: Visibility Culling using Hierarchical Occlusion Maps. In: *SIGGRAPH 97 Proceedings, Annual Conference Series, 77-88,* 1997.



(a) replicated trees ($1.4 \cdot 10^{14}$ triangles), per pixel reconstruction, rendering time: 6.1 sec.

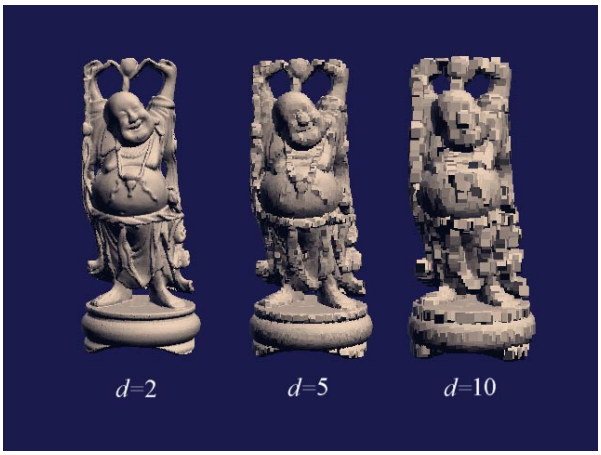


(b) replicated trees ($1.4 \cdot 10^{14}$ triangles), per pixel reconstruction, average over 10 images, total rendering time: 93.7 sec.

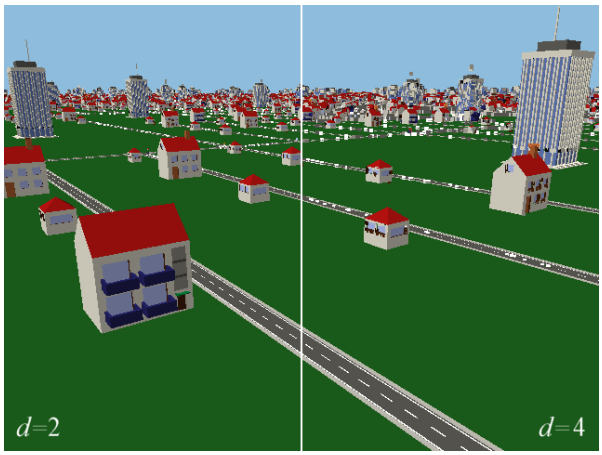


(c) landscape scene ($4.1 \cdot 10^{8}$ triangles), per-pixel reconstruction ($d = 1$), phong illumination model, rendering time: 19.2 sec

(d) landscape scene ($4.1 \cdot 10^{8}$ triangles), splatting, $d = 5$, diffuse illumination model, rendering time: 0.64 sec



(e) "Happy Buddha" mesh (10^{6} triangles), splatting with different splat sizes d, rendering times: 438, 96, 34 msec



(f) city model ($3.4 \cdot 10^{6}$ triangles), splatting with different splat sizes d, rendering times: 560, 225 msec

Figure 14: example renderings