Stylization-based ray prioritization for guaranteed frame rates

Bernhard Kainz, Markus Steinberger, Stefan Hauswiesner, Rostislav Khlebnikov and Dieter Schmalstieg *
Graz University of Technology

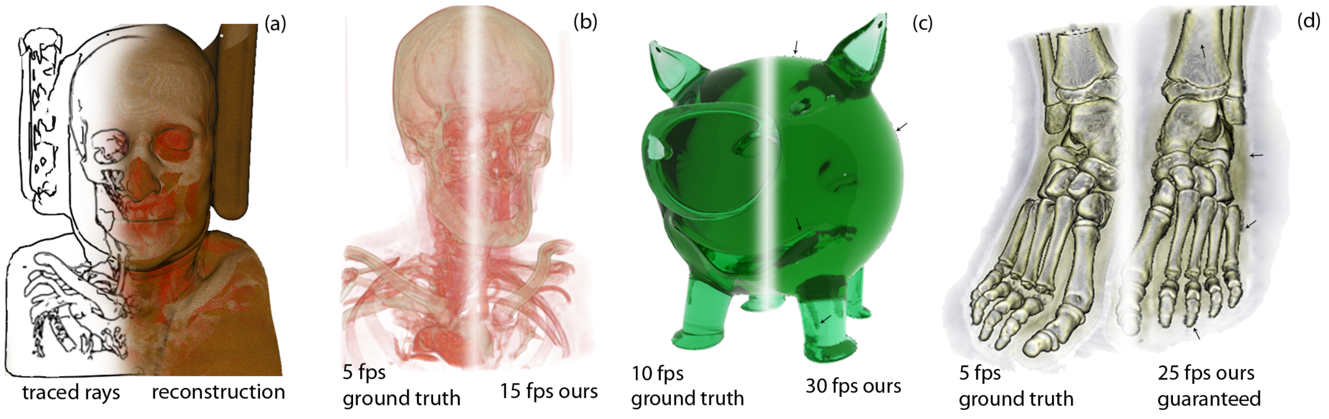

Figure 1: *Four different scenarios rendered with a conventional ray-based rendering engine and with the presented adaptive approach in a* 1024×768 *viewport. (a) shows the areas in which the rays are fully computed (left) and the resulting reconstruction by our algorithm (right). (b) shows how our approach can boost the frame rate while preserving the quality level. Our algorithm can also be applied to ray-tracing scenes with complex materials (c). Priority sorting of the expected visual quality of a pixel allows us to guarantee frame rates for every type of ray-based environment while maintaining a visually appealing result (d). Selected non-obvious artifacts, which are introduced by using our method for guaranteed frame rates, are marked with small arrows in (c) and (d).*

Abstract

This paper presents a new method to control graceful scene degradation in complex ray-based rendering environments. It proposes to constrain the image sampling density with object features, which are known to support the comprehension of the three-dimensional shape. The presented method uses Non-Photorealistic Rendering (NPR) techniques to extract features such as silhouettes, suggestive contours, suggestive highlights, ridges and valleys. To map different feature types to sampling densities, we also present an evaluation of the features impact on the resulting image quality. To reconstruct the image from sparse sampling data, we use linear interpolation on an adaptively aligned fractal pattern. With this technique, we are able to present an algorithm that guarantees a desired minimal frame rate without much loss of image quality. Our scheduling algorithm maximizes the use of each given time slice by rendering features in order of their corresponding importance values until a time constraint is reached. We demonstrate how our method can be used to boost and guarantee the rendering time in complex ray-based environments consisting of geometric as well as volumetric data.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; I.3.8 [Computer Graphics]: Applications—;

Keywords: ray based image generation, line features, guaranteed frame rates

1 Introduction

A common challenge of high-quality ray-based image generation is maintaining the interactivity of the applications. This interactivity is normally achieved by sacrificing some of the image quality during the interaction and by progressively refining the result as soon as the scene interaction stops. The simplest method in this context is regular sub-sampling: rendering the scene in a small viewport during interaction and stretching the resulting image to the target image size using linear interpolation. This method indiscriminately discards features and results in a blurred render frame or block artifacts.

Adaptive sampling approaches try to assign the computational costs to regions with high image fidelity and to approximate the remaining image parts. Typically, these techniques use features that have been detected in the image plane. These approaches obviously require the final result as an input for the optimal result, which is impossible. Hence, image regions from previously rendered frames [Dayal et al. 2005] or sparsely sampled regions [Painter and Sloan 1989] are used. However, image space methods suffer from less accuracy than object space methods because of the required projection to discrete image pixels. Furthermore, many image space algorithms may cause more computational overhead than benefit because of the already high GPU utilization of modern ray-based image synthesis systems [Parker et al. 2010].Therefore, we investigated the key element of adaptive approaches, which is the determination of which elements of an object can be coarsened and which must be preserved. Much perceptually based research has been performed in this area by researchers from the Non-Photorealistic Rendering (NPR) community. However, these results have only been used for scene stylization and enhancement. We present a new sampling strategy for ray-based image synthesis, which uses information about object space features that are known to support the comprehension of 3D shapes [Cole et al. 2008]. In this paper, these NPR techniques are used to control the reduction of ray samples and thus to achieve a higher image quality while maintaining the same level of interactivity.

Our implementation produces a feature buffer for every frame that

*kainz|steinberger|hauswiesner|khlebnikov|schmalstieg@icg.tugraz.at

is efficient enough for use during the ray generation as a lookup table for the required ray density. We derived a feature priority map from the feature buffer that consists of silhouettes, suggestive contours, ridges and valleys, all of which affect the ray density differently (see Figure 1(a)). Because different features generate different ray densities, our method is able to support an importance-driven rendering to guarantee the minimum desired frame rate. Even though our main focus is the visualization of volumetric datasets, we also demonstrated a way to apply our method to geometric objects with highly complex materials in ray-tracing scenes. Figure 1 shows some selected examples using this approach. The main contributions of our method can be summarized as follows:

- A method that allows the optimization of the ratio between the sampling rate of the scene and its resulting perceptual quality (Section 4).
- A progressively refineable sampling pattern, which is used to reconstruct sparsely sampled regions of the image without the need for frame-to-frame coherence (Section 4.4).
- An algorithm that uses our method to guarantee frame rates while maximizing the visual quality within the available time frame (Section 5).
- An evaluation of different object space line features to categorize them based on their abilities to enhance the image quality (Section 6).
- A discussion of an addition to our method, which uses a fast computation of the image space visual saliency. With this method, we can also include features that can only be evaluated in image space (e.g., textures and shading details) (Section 9).

2 Previous work

Previous researchers have been concerned with the real-time performance of ray-based image generation algorithms. Recent work has introduced the exploitation of modern GPUs for solving the brute-force full-resolution ray traversal interactively, while coarse adaptive and progressive sampling approaches have been discussed since ray-tracing algorithms first became available. We give a brief overview of recent GPU methods and adaptive progressive rendering methods in Section 2.1 and discuss possible scene feature computation strategies in Section 2.2. A further overview of the reconstruction techniques for sparsely sampled data is given in Section 2.3, and the attempts to guarantee a minimal frame rate are outlined in Section 2.4.

2.1 Interactive ray-based rendering

Exploiting the GPU. Numerous rendering engines have been developed to deal with one of the most computationally expensive problems of computer graphics: ray-tracing. Besides CPU-based libraries [Parker et al. 1999; Wald et al. 2007], most recent GPU approaches reach remarkable frame rates for low- to medium-complexity scenes [Seiler et al. 2008; Parker et al. 2010] in full quality. However, screen filling scenes or scenes with high complexity are still too slow for hard real-time constraints.Furthermore, rendering algorithms that aim at achieving real-time performance for the full-quality ray-casting of volume data use empty-space skipping [Li et al. 2003], iso-surface ray-casting [Wald et al. 2005; Wang and JaJa 2008], ray pre-integration [Engel et al. 2001], homogeneous region encoding [Freund and Sloan 1997] and many kinds of direct GPU implementations [Fernando 2004, Chapter 39].

Adaptive progressive rendering. Adaptive approaches, such as the one presented in this paper, aim instead for the best possible trade-off between interactive frame rates and the loss of image quality instead of finding the maximum achievable frame rate for a full quality image. Finding this trade-off is still an ill-defined problem because the perception of quality differs between human beings and between applications. However, several algorithms exist to accelerate rendering speeds through ray reduction. The simplest method is a regular sub-sampling with a nearest neighbor interpolation. As discussed by [Mitchell 1987] and still used in many interactive ray based rendering systems [Schroeder et al. 1998; Wald et al. 2002], this method is prone to strongly perceivable aliasing artifacts during the interaction. To deal with this problem, most related work has investigated the impact of different sampling pattern strategies in image space [Dippé and Wold 1985; Painter and Sloan 1989; Notkin and Gotsman 1997]. The sampling pattern is usually visually noticeably refined over time until a desired quality level is reached.

Levoy reformulated the front-to-back image order volume rendering algorithm to use an adaptive termination of ray-tracing [Levoy 1990]. The subdivision and refinement process is based on an ϵ threshold and does not consider human feature perception and temporal coherence. Later work altered the ray termination criteria [Danskin and Hanrahan 1992] depending on the required rendering time or used texture-based level of detail [Weiler et al. 2000], topology guided downsampling [Kraus and Ertl 2001] or multiple resolutions of the same dataset [La Mar et al. 1999; Boada et al. 2001].

2.2 Important image areas

The choice of a suitable sampling pattern is crucial for adaptive rendering. For non-trivial systems, the pattern refinement strategy is usually chosen depending on prominent features. In the following paragraphs, we discuss our selected methods to find those regions.

Image space methods. Most methods refine the image sampling pattern based on image intensity variances. Early algorithms assume that image areas with high frequencies require a denser sampling than do large uniform areas [Lee et al. 1985; Painter and Sloan 1989] to gain a visually acceptable result. Later systems adapt this assumption towards the limitations of the human visual system. [Ramasubramanian et al. 1999] have been one of the first who have successfully introduced an image-based perceptual threshold map which steers the sampling density of a global illumination path tracing algorithm. The Ramasubramanian system shows that it is possible to generate images, which have visually no difference to a ground truth, with only 5-10% of the rays which have been used for a reference solution.

During the development of our algorithm, we have also experimented with the extraction of features from image space by using visual bottom-up saliency [Itti et al. 1998].The saliency of an image is usually defined as a measure of how much a particular location contrasts with its surroundings in dimensions such as color, orientation, motion , and depth. The resulting feature classifications are very similar to the object-space results but at a lower performance and accuracy, and they must be computed for every frame. For these reasons, we first chose the object-space approach of analyzing meshes. In Section 9, we discuss the incorporation of saliency computations for scenes that show mainly textured objects or hard shadows and strong reflections. One main problem is that for an optimal result, the final image would be needed at full resolution before the scene is rendered. However, a full resolution image is only available from previous frames, which can be used to generate a spatiotemporal gradient cross-hair as introduced by [Dayal et al. 2005]. This approach introduces severe spatial and temporal noise.

However, the core of our method does not require any temporal coherence.

Non-Photorealistic Rendering of line features. NPR deals with salient object features, often directly in object space. Related rendering techniques are mainly used for illustrative rendering and in cognitive science. [Cole et al. 2008], for example, to show that object contours including the object silhouette are also used by artists to outline scenes. Several visualization algorithms show that these features can be used to simplify complex scenes for a better understanding of the essential parts [Bruckner 2008; Burns et al. 2005; DeCarlo and Rusinkiewicz 2007].

The features of an object or a scene can be extracted in various ways: meshes can be analyzed in object space or, after rendering, in image space. The same method applies to volumetric data sets, where the object space contains a voxel grid instead of a set of geometric primitives. Finding features in a rendered image has the advantage of including textures and other effects, while in object space, more accuracy is usually available because the data have not been discretized into pixels. Moreover, the features in object space may be view-independent, which allows their reuse without recomputation. Figure 2 gives a visual impression of some sparse object features that we evaluate for ray decimation in this work. Our definitions of object features are similar to those from [DeCarlo and Rusinkiewicz 2007].

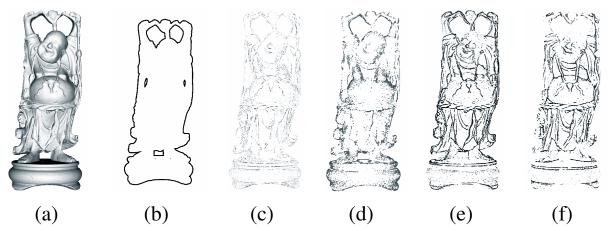

Figure 2: *The happy Buddha object (a) rendered with different sparse line features. Silhouettes (b), suggestive contours (c), suggestive highlights (d), ridges (e) and valleys (f) are evaluated for ray decimation in this work.*

2.3 Sparse data reconstruction

Computing only rays for important areas means that the final image has to be reconstructed from those sparse samples. Numerous approaches exist besides the simplest, conventional approach of regular subsampling. This attempt requires a nearest neighbor computation or a linear interpolation and leads to perceivable block artifacts. A good general overview of non-homogeneously sampled data reconstruction is given in [Amidror 2002].

Specialized approaches for computer graphics can be found for point-based rendering. The widely used *pull-push algorithm* [Gortler et al. 1996] utilizes a pyramid algorithm for surface reconstruction. It has been adapted for the image-space reconstruction of under-sampled point-based surfaces by [Grossman and Dally 1998]. [Pfister et al. 2000] extended this approach to fill the holes between splats. Unfortunately, these approaches are not suited for direct GPU implementations, as stated by [Marroquim et al. 2008]. The approach of [Marroquim et al. 2008], who proposed a GPU implementation for large point-based models with elliptic box-filters and deferred shading, is also applicable to the reconstruction problem in this work. However, its computational overhead is still higher for large viewports than that of the method presented in Section 4.4.

2.4 Guaranteed frame rates

To the best of our knowledge, our method is the first that successfully implements an algorithm to guarantee a certain minimal frame rate and still maintains an acceptable image quality for ray-based image generation. [Pomi and Slusallek 2005] proposed that a guaranteed frame generation time would be essential for mixed reality TV studio ray tracing applications, but they did not implement such an approach. For non-ray-based rendering approaches, a few systems that guarantee a certain frame rate exist. For example, [Jeschke et al. 2005] replaces complex objects optimally by impostors. These examples show that several applications require guaranteed frame rates. However, this problem is not well researched.

3 Overview of the method

Our approach consists of two passes. First, a set of line features is extracted from the data set's corresponding meshes and projected to the screen space, resulting in a feature buffer. According to our evaluation (see Section 6), we assign priorities to different types of features. The feature buffer is evaluated during the ray setup and traversal, which forms the second pass. Given that we can assign a priority value to every ray, it is possible to construct a rendering system that aims at producing the best image quality within a given time frame as outlined in Figure 3.

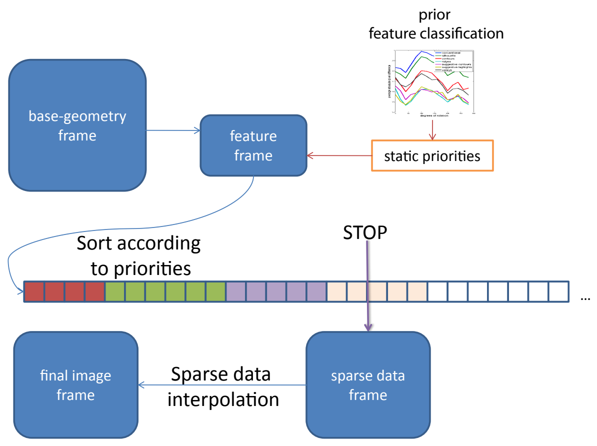

Figure 3: *Overview of our prioritized rendering algorithm. Every ray's priority is computed according to Section 4.2 and is used to sort the pixels in a one-dimensional priority queue. If a certain time limit is reached, the rendering process stops.*

We adaptively adjust the image space sampling frequency according to the feature buffer. More rays are sent into the scene in feature-rich areas and their vicinity, while the sampling frequency for feature-poor areas is strongly decreased. The same strategy can be used for per-ray quality parameters (such as ray bounces, object space sampling frequency or stopping threshold). Finally, we reconstruct the image by filling in color values for pixels to which we have not previously assigned a ray. In Section 4.4, we present a suitable method for a full image reconstruction using a fractal reconstruction pattern and an adaptive linear interpolation.

Our method is applicable to a ray-based rendering of geometric surface meshes and to volumetric data sets. The only difference is given by the feature extraction step. For surface meshes, the feature-forming geometry is defined by the mesh itself. Using volumes requires the extraction of multiple iso-surfaces based on an evaluation of the given transfer function before the line features are rendered.

4 Importance-driven sampling and reconstruction

To control the frequency of the sampling pattern based on different types of features, we render an importance buffer in each frame (Section 4.1). This importance buffer is filled by a projection of each single line element to the screen space followed by a computationally cheap falloff estimation. This buffer defines the importance of pixels (Section 4.2). We derived the corresponding sampling density using a fractal sampling pattern (Section 4.3).

4.1 Object space importance buffer

To provide a sufficiently high frame rate in the first render pass, we have extended the approach from [DeCarlo and Rusinkiewicz 2007] with selective GPU acceleration techniques, which are described in Section 7. For our method, the feature projection buffer must be produced in a time frame that is shorter than the savings from the main rendering pass. In practice, we can render this step at several hundred frames per second because the features are rendered as simple OpenGL lines. These lines are also reused from frame to frame, and the vertices are only either removed or inserted. To simulate smooth features and to gradually decrease the priority in the vicinity of features, we can replace these lines by textured triangle strips, compute a distance transform on the feature buffer, or use a fractal pattern as described in Section 4.4, the last option providing the highest flexibility. In this way, we generate an intensity falloff around the features. To remove any hidden features, we also render the underlying base mesh with a homogeneous white surface and a fully opaque one. Next, we encode the resulting feature projection buffer as follows:

- Red pixels define a pure background,
- Black pixels define primary salient features, (*luminance = 0* $\widehat{=}$ *contours*)
- Gray to black pixels define secondary salient features, (*1.0 - luminance = feature priority* $\widehat{=}$ *ability of the feature to improve visual quality* as defined in Section 6)
- White defines homogeneous object areas with no salient features.

4.2 Adaptive subsampling

When the feature frame is available, the actual ray traversal starts. Every pixel of the output frame defines a possible ray starting point which is equal to one thread in terms of GPU stream processing. If the feature frame contains a red pixel at the thread's position, this ray's thread will immediately return and fill its corresponding position in the output buffer with the background color. If the feature frame contains a pure black pixel, i.e., a primary feature, the ray will be processed completely until the given convergence criteria are fulfilled. If the feature-buffer shows a secondary feature, we consult an adjustable ray priority table, which is used to determine the image space sampling pattern (see Section 4.3). The combination of feature priority p_{feat}, which is deduced from the intensity of the feature buffer, combined with the pattern priority p_{patt}, which is read from the sampling priority table, is used to determine the ray's priority p_{ray}. Every arbitrary function f_{map} with two input parameters is possible to combine these two independent priorities:

$$p_{ray} = f_{map}(p_{feat}, p_{patt}) \tag{1}$$

For an efficient implementation, we use a second-order Taylor series approximation, because its implementation consists only of basic and fast algebraic operations. It is defined by six fixed parameters $\alpha_{i,j}$:

$$p_{ray} = \sum_{i+j \leq 2} \alpha_{i,j} \cdot p_{feat}^{i} \cdot p_{patt}^{j} \tag{2}$$

The rays are traced according to their priority p_{ray}. If the mapping function captures every ray's contribution to image quality, a fixed threshold can be used to only trace rays with a high contribution. In this case, $-\alpha_{0,0}$ can be used as the threshold and rays with a priority above zero are traced. Another option to sort rays according to their priority and use the available time slot to draw the rays with the highest contribution (see Section 5). The way that f_{map} and thus the α values are chosen, controls the influence of the feature priorities on the output image. Low weights for terms dominated by p_{feat} will result in a nearly uniform pattern, while a high contribution of p_{feat} creates samples at only the feature areas. A good trade-off between these extremes is a method that creates a dense sampling pattern along important features while reducing the number of samples along the transition from a feature to feature-less areas. Lower priority features would thus receive a lower sampling density than would higher priority features, and homogeneous areas would contain only a few sampling points (see also Figure 7).

In practice, we have used a rather simple choice for the α values: $\alpha_{2,0} = \alpha_{0,2} = 0$ and $\alpha_{i,j} = 0.5 \pm 0.2$ for all remaining terms. However, an optimal mapping function f_{map} takes information about the rendered objects into consideration. Images of strongly transparent objects naturally show few homogeneous areas; thus, increasing the influence of p_{patt} (increasing $\alpha_{0,1}$ and $\alpha_{0,2}$) will have a positive influence on the image quality. Nearly opaque objects with low color variation will benefit from an increasing influence of p_{feat} (increasing $\alpha_{1,0}$ and $\alpha_{2,0}$), as most variation in color appears along the feature regions.

For the remaining case – a white pixel in the feature buffer, which indicates no salient feature at that position – we assign a feature priority of zero and only use the ray pattern priority to determine whether the ray should be traversed. All non-background rays, which have not been traced, are subject to a reconstruction step as described in Section 4.4.

4.3 Sampling pattern

The choice of an incrementally refineable sampling pattern is crucial to smoothly add detail to transitions between fully traced areas and a coarsely traced background. The design of this sampling pattern should further consider the possibility of interpolating the resulting ray pattern efficiently. Both problems can be addressed by defining a fractal sampling scheme that considers only two local shapes: a square and a diamond (45° rotated square). We start with the coarsest sampling density, which only needs to be sufficiently coarse, and a power of two, and create a square pattern by placing a ray at every 8×8 square of pixels. We then place a sample at the center of the square that exactly matches the center of a pixel and splits every square into four triangles. Together with the surrounding squares, which are augmented with an additional sample, a diamond pattern results. The density of this pattern can be increased by placing a ray at the center of each diamond. This procedure leads again to a uniform square pattern. Repeating these steps places rays at exactly the centers of pixels until every pixel is covered with a single sample. The associated priority values are deduced by starting with the maximum priority and linearly decreasing the priority with each new shape. The whole procedure is outlined in Figure 4.

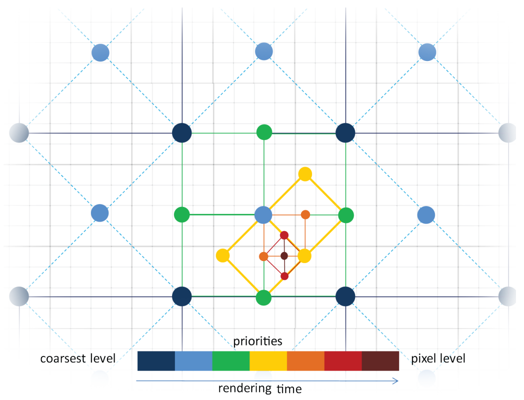

Figure 4: *We use a sampling priority pattern similar to that outlined in this figure. For illustration reasons, this figure shows a much finer grid than would be used in reality. Blue defines the initial rays with priorities 1.0. With every sampling step, the pattern becomes finer, and the priority decreases (color coded in the figure).*

The pattern can be refined locally and thus increase the sampling density for arbitrarily sized regions.

4.4 Image reconstruction

To improve the quality of the reconstructed image, we have investigated methods to fill areas for which no rays have been traversed. We focus on reconstructing without losing much performance. Our approach linearly interpolates samples based on the fractal pattern presented in Section 4.3.

Various methods exist for interpolating non-homogeneously sampled data [Amidror 2002]. However, our sampling pattern allows us to combine the choice of ray locations with their interpolation and compute both steps efficiently. A pixel contained in the interior of a square pattern can be constructed with a bilinear interpolation from the four anchor points defining the square. Because the diamond shape is a rotated square, we only need to rotate the pixel position accordingly to enable a standard bilinear interpolation for this shape.

The transition from one interpolation density to the next requires an additional step, as up to three anchor points might be missing. In this case, we have to interpolate the missing anchor points from the coarser pattern first. From another point of view, we add a ray according to the pattern described in Section 4.3, but instead of tracing it, we interpolate its value from the already given rays. As this new anchor point is placed exactly in the middle of the already existing ones, the linear interpolation breaks down to an evenly weighted mixture of the four anchor points. For an efficient implementation, we have to make sure that we do not create a dependency chain when gradually decreasing the sampling density. If the sampling density is decreased too abruptly, there will not be enough lower density rays available to construct the missing points for the next higher density. Thus, the missing point has to be constructed from another density level first. Therefore, for maximum performance, we make sure that the features are smoothed sufficiently such that enough rays are traced to construct all of the missing anchor points in a single step.

5 Guaranteed frame rate rendering

For a continuous rendering scenario with a good frame-to-frame coherence, we can build a reactive rendering system that adjusts the number of traced rays depending on the time needed for previous frames. This step is possible by dynamically adjusting the threshold that defines which rays shall be rendered, i.e., by decreasing $\alpha_{0,0}$ by a fixed value if the frame rates are too low. If the scene is static and the camera is still, the rays traced in the previous frame are reused, and we progressively add new rays by increasing $\alpha_{0,0}$. Thus, the image converges to the highest quality.

For hard real-time scenarios with low frame-to-frame coherence or with little still image renderings with progressive refinement over time, the aforementioned approach fails. An unexpected load on the GPU, complex objects popping up in the scene or the simple lack of a previous frame prohibits us from deducing enough information for the next frame. However, in these cases, we can still rely on the ray priorities to guarantee the given update rates. We require an additional sorting step before the actual ray tracing is conducted. Every ray's priority is computed according to Section 4.2 and inserted into a one-dimensional priority queue. In this scenario, the parameter $\alpha_{0,0}$ is irrelevant because it has no influence on the sorting order. During the following rendering step, each block of threads fetches a set of rays from the front of the queue and processes them. This step is repeated until the available time frame is nearly over. The use of this priority queue guarantees that the available time is spent on rays that have been classified as being the most important. For the sorting itself, we use a fixed number of buckets instead of completely sorting the queue to increase performance. As we cannot guarantee that all elements within a bucket will be processed, we randomize the order in every bucket. Otherwise, the render order for similar ray priorities would match the insertion order, and the sampling density might thus only increase locally.

Our experiments have shown that the reconstruction step's execution time is very short and has little variation. We can thus measure an upper bound for this step in the initialization phase and reduce the time frame during the rendering accordingly to have enough time for the reconstruction step. This setup enables us to output a frame within the desired latency. The time measurement is performed on the graphics card itself, which allows each block of threads to work autonomously without synchronization via the host. For static scenes, we can again use our system for a progressive rendering. As low priority rays are still present in the queue after the time frame is over, we can simply re-launch the rendering kernel right after presenting the current quality level. In this way, the next set of rays are traced within the next time frame, and we are able to progressively update the scene with the next lowest ray importance level.

6 Feature Classification

To evaluate how features improve the visual quality of the result, we have tested ray traced objects and selected volume-rendered objects with different transfer functions. We assume as the lower visual quality bound the simplest subsampling approach: rendering at a lower resolution with a block filter kernel reconstruction. The upper visual quality bound is given by the ground truth (ray tracing at full resolution). We evaluate subsequently how the image quality improves when pixels, which are marked as a certain feature, are rendered with an increased sampling density.

To quantify the visual improvements of different features, we use two image comparison metrics (comparing the feature-enhanced image to the ground truth image): the average absolute pixel difference (AAPD) and the Structural Similarity Index (SSIM).

The first quantity is simple, parameter-free, and easy to compute. Moreover, it has a physical meaning similar to the mean square error (MSE), the energy of the error signal. Problems with averaging methods arise when the subjective human's perception of image quality has to be quantified. As it was shown in [Wang and Bovik 2009], these methods are very similar despite the differences in image distortions.

The second quantity, SSIM, is a generalized form of the Universal Quality Index and was proposed by [Wang et al. 2004]. This metric shows distinct values for different kinds of image distortions according to well-defined luminance, contrast, and structure comparison measures.

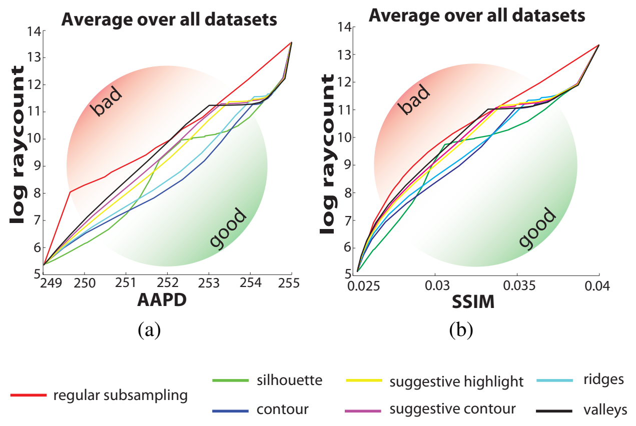

Figure 5: *These graphs evaluate the render quality in terms of the AAPD (a) and the SSIM (b) compared with the required logarithmic ray count that is needed to reach that quality. Both plots are an average of our test datasets (ray-traced geometry as well as volumes). For both evaluation methods the center areas of the graphs show evidence that the conventional regular subsampling approach performs worst, whereas rendering along contour lines performs best.*

Figure 5 shows that all object space features improve the image quality during interaction, compared with the conventional regular subsampling rendering. We observed that using the exterior silhouette leads to the best result for objects with a low interior feature count. Otherwise, using ridges and contours yields the best outcome. In the case of objects with a clear boundary (e.g., as used for ray-tracing), the external silhouette alone shows the highest relative quality improvement. For translucent volumetric objects, the external silhouette does not necessary improve the image quality because it might cover the whole dataset and exclude the (probably more important) internal features. Based on our observations from automatic tests, we first roughly classified the features into *strong visual features* and *medium visual features*. We did not introduce the class *weak visual features* here because we have already considered regions on/in an object with no response to any feature extraction algorithm as regions with a low image signal frequency. Our experiments show that *contours* in particular lead to a much better relative perception during the scene interaction for our tested scenes. Ridges and (subjectively) *exterior silhouettes* also yielded a good result in all of the tested scenes, especially for transparent volumetric objects. Therefore, we categorize *contours* as *strong features* (along with the *external silhouettes* as a subset of *contours*) and the remaining ones as *medium features*. In Section 5, we directly use the results from Figure 5 to define static feature priority lookup tables for each separate feature in a numerical way. Using the values measured in this section, we can provide good default values for the priority lookup tables. However, a user is still able to alter these priorities in our system.

It is desirable to evaluate the importance of different features for every frame independently. For a fully automatic evaluation of the image quality improvement of different line features per frame, obtaining the ground truth is necessary, which is of course not feasible during runtime. Therefore, users can alter the proposed feature ranking in our system during runtime. Preliminary experiments with this feature have shown that users tend to fully disable features such as suggestive contours, suggestive highlights and valleys to gain higher frame rates. These features have also shown a low visual improvement during our offline evaluation as outlined in Section 6.

7 Implementation

We have implemented our method as part of the OptiX SDK [Parker et al. 2010], which we have enhanced with volume rendering abilities. OptiX provides a C++/CUDA-based programming interface, which is specialized for ray-tracing applications. Because several materials, like the glass effect, which has been used in this work, are already implemented in the SDK, we only had to extend the framework to include a volume material and specialized ray generation programs (*cameras*) to support our method.

In OptiX, a simple ray-tracing program normally consists of a combination of a *hit function*, a *trace function*, a *miss function*, and a *camera* for the ray setup. The main functions are executed per ray. The *hit function* is used to intersect rays with object surfaces in the scene. The *trace function* evaluates the color contribution of a ray between two intersections, and the *miss function* fills rays that hit no geometry with a defined background color. These functions are implemented in separate CUDA files, which are preprocessed by the OptiX SDK.

7.1 Feature preserving adaptive sampling camera

The rays are set up by a ray generation program, which can be seen as a *camera*. We use a pinhole camera model as the basis for our implementation and alter the ray generation scheme by our adaptive approach. The feature frame is rendered by using an extended version of the publicly available framework provided by [DeCarlo and Rusinkiewicz 2007], which is based on the Princeton *Trimesh2* library. To provide high frame rates for very complex objects, we extended this library with GPU-accelerated calculations. Therefore, we moved all per-frame calculations (e.g. $\hat{n}(p) \cdot \hat{v}(p)$) to CUDA kernel functions and attached the line-output to an OpenGL Vertex-Buffer object. This vertex buffer is subsequently rendered into an *OpenGL Framebuffer object*, which is concurrently mapped as a texture in the OptiX context. The ray setup is then done according to this texture as described in Section 4.2.

For volume rendering, we have also attempted a direct feature line extraction as proposed by Burns et al. [Burns et al. 2005]. Experiments with the Burns system have shown that the frame rates are not as high as those obtained with iso-surfaces, which are used in the DeCarlo system, especially for very large volumes and multiple transfer function peaks. This fact can be explained by the difference in the order of complexity when processing a surface ($O(n^2)$) compared with when processing a volume dataset ($O(n^3)$). Because iso-surfaces for the feature frame generation only have to be recalculated when the transfer function changes, we have decided to use the feature extraction approach from De Carlo et al. However, it would also be possible to use the approach of Burns et al. because it also shows frame rates that are high enough to meet our two-pass rendering criteria.

7.2 Volume rendering

We have implemented a standard ray-casting approach as a *material trace function*. In contrast to tight-fitting bounding geometry volume rendering systems, the volume bounding geometry can be a simple cube or a sphere instead of a tight fitting one. This detail reduces the necessary intersection calculations and maximizes the thread coherence, which was stated by [Parker et al. 2010] to be one of the most important factors for an efficient execution. Because every ray can store a certain amount of payload, we save the entrance point and the exit point of the *hit function* in every ray's payload structure. After transforming these two points to the volume object space, we let every ray accumulate all of the values in between, depending on the given transfer function. In addition, the values are shaded according to the Phong illumination model depending on the approximated volume gradient.

Mesh extraction Our method requires a smooth surface mesh for feature rendering. At that stage, we distinguished between a pure geometric input for ray-tracing applications and volumetric data sets for a direct volume ray-casting. In the first case, the input mesh can be used directly by the feature extraction step. The second case requires an intermediate step depending on the used volume transfer function and the volume histogram. In our case, a transfer function is defined by several color gradients, mapping a certain intensity range to a defined color and opacity. One can also define high dimensional transfer functions for a better visual result (e.g., using the gradient magnitude as a second dimension, as proposed by [Kniss et al. 2001]). However, for an iso-surface extraction, the peak value of the direct (1D) mapping gradients or the projection of the color gradient's opacity peak to the intensity axis for high-dimensional transfer functions together with the peak of the volume histogram is sufficient. Consequently, we define the necessary iso-values for a multi-iso-surface extraction as the highest peaks of the volume's histogram if the transfer function is not zero at that position. We use a fast GPU-accelerated marching cubes implementation (CUDA version of the [Lorensen and Cline 1987] algorithm) to extract the iso-surfaces whenever the transfer function changed. Because the resulting meshes are over-tessellated, we simplify this mesh with a GPU-based simplification method as described in the following paragraphs.

Mesh preparation The quality and size of surface meshes, used as inputs for our algorithm, vary dramatically. Ray-tracing applications are often applied to high quality meshes with hundreds of thousands of triangles. The iso-surface meshes extracted from volumes are known to be noisy and often contain lots of small triangles, which can be merged without a loss in quality. We thus use a combination of mesh smoothing [Taubin 1995] and mesh simplification [Luebke and Erikson 1997] to generate meshes that fulfill our demands: (a) the mesh contains little noise, (b) the number of faces is low enough to generate the feature buffer quickly, and (c) main features from the original mesh are conserved at the according position.

Our algorithm successively applies Taubin smoothing, mesh simplification and another instance of Taubin smoothing. The first smoothing step is especially important for iso-surfaces extracted from a volume. Taubin smoothing preserves the volume of the mesh and thus also conserves the location of remaining features. Our implementation of the mesh simplification method is run once per mesh as a preprocessing step and does not include any view-dependent simplifications. A static simplification based on a fixed error metric turned out to be sufficient for our demands. Both algorithms allow a highly parallel GPU-based implementation, which enables low latency on input data changes. The overall process takes up to a second, depending on the mesh complexity and the number of peaks in the transfer function.

8 Results

In Section 6, we present our results on how much a certain object feature can improve the visual quality. In this section, we evaluate the performance of the overall system. Our test system is equipped with an Intel i7 Processor, 6 GB System memory and an Nvidia Quadro 6000 graphics card. Figure 6 shows the increase of the AAPD (a) and the decrease of the SSIM (b) for increasing guaranteed frame rates.

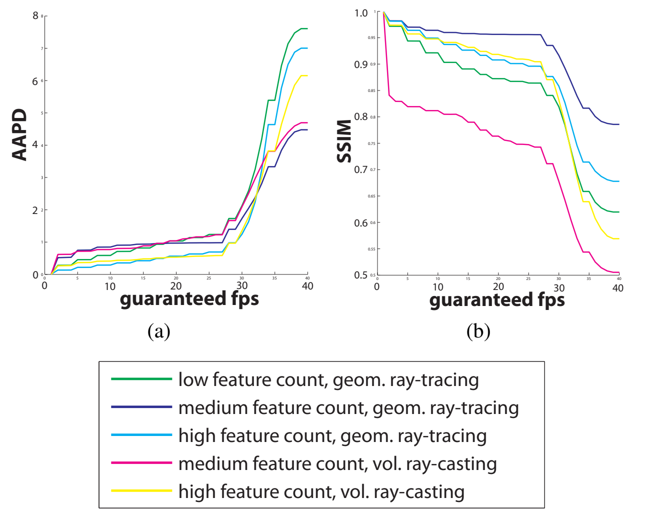

Figure 6: *These plots show the increase of the AAPD (a) and the decrease of the SSIM (b) for increasing guaranteed frame rates averaged over different data sets. The viewport for this test was* 1024 × 768. *Note that, after a certain guaranteed frame rate, the time frame becomes too short to render the important features. This issue becomes apparent in the plots by the bend at approximately 35 fps.*

The feature lookup texture can be rendered with up to 1000 frames per second on a modern graphics workstation in a moderate viewport and it does not need to be of the same size as the render frame. Therefore, the computation time for this step can be neglected. For quality estimation, we use the fractal pattern interpolation image reconstruction method, as described in Section 4.4. Table 1 gives an overview of the overhead computation times of our method compared with the unaltered ground truth. Figures 7 and 8 show the decrease of quality with an increasing frame rate demand. The image quality remains stable as long as the frame rate is reasonably adjusted. Figure 7 also shows the pixels that are required to calculate a full ray traversal and compares our image reconstruction method to a regular subsampling with linear interpolation. Figure 8 shows the quality decrease for a volumetric object.

9 Image space saliency

Because pure object space features do not incorporate high frequencies in textures and hard shadows or reflections/refractions, we carried out additional experiments to enhance the feature-buffer with this information. If required, users can activate the saliency information for the features buffer. Therefore, we rendered the unaltered scene at a lower resolution and calculated the saliency of this image before the feature frame is generated. For that step, we make use

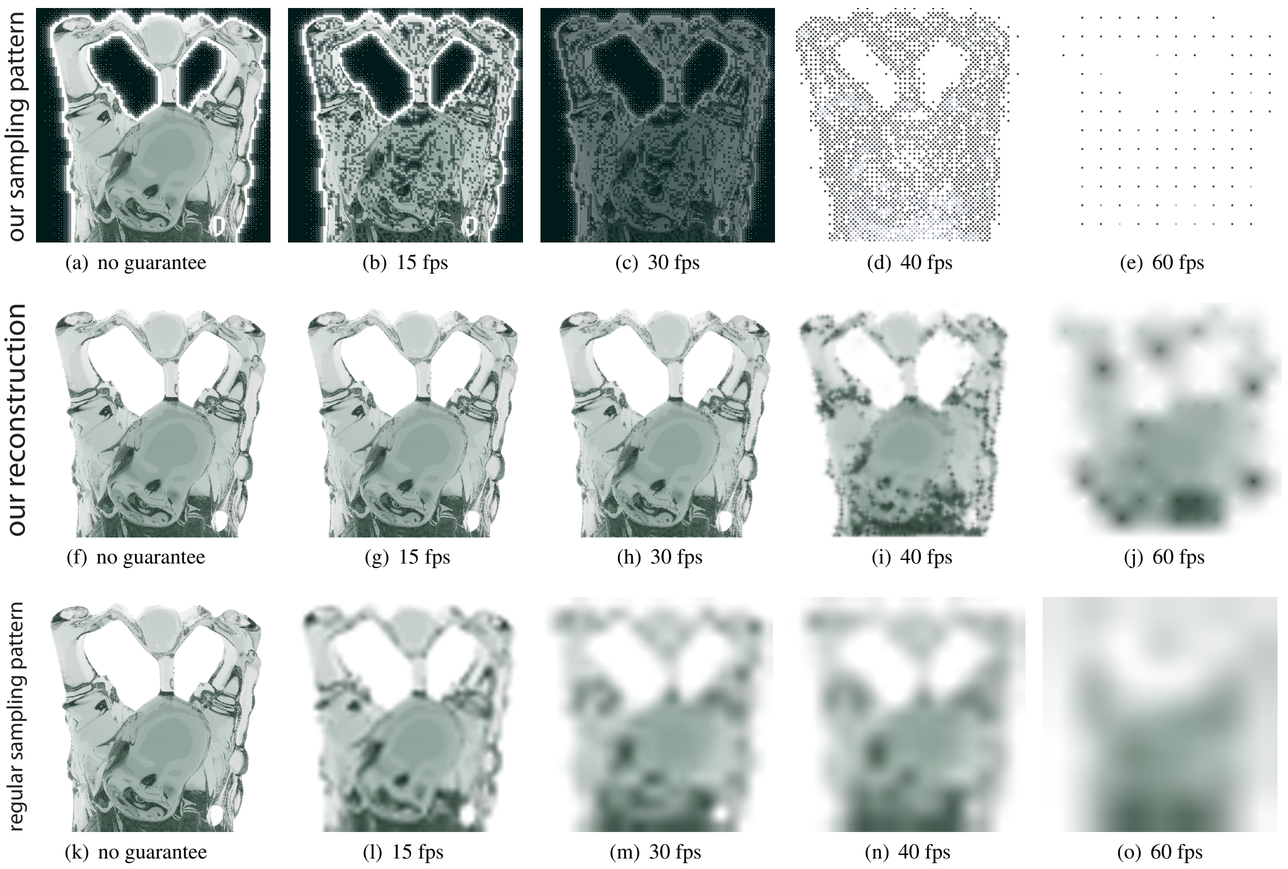

Figure 7: *This figure illustrates the decreasing ray count with increasing requested guaranteed frame rates for a* 1024×768 *viewport. The top row shows the actual pixels that have been traced, and the middle row shows the result with our fractal pattern interpolation scheme. (i) and (j) of the top row are the edge images of the traced pixels to emphasize their positions in the printed versions of this paper. The bottom row shows the results using a regular sub-sampling pattern with a linear interpolation for comparison.*

of the OptiX rendering engine's ability to render small scenes at full resolution with very high frame rates. If the image space addition is selected, a third render-pass is added in the beginning of our method, which rendered the unaltered scene at a very low resolution (typically a factor of 4 to 8 smaller, depending on the graphics hardware used and target resolution). Subsequently, a GLSL shader implementation of the method from [Itti et al. 1998] is applied to compute the image's saliency. The feature buffer is then enhanced with a linearly interpolated version of this image. The saliency information defines some additional lower levels within our priority queue. This step usually requires 10-20 ms. However, for this step, the required rays are reused in the reconstruction step to further improve the image quality. As an example, the feature buffer including the proposed saliency measure is shown together with the resulting reconstruction in Figure 9.

The visual saliency information of a small scene viewport is only a rough approximation of the full resolution; hence, we assigned the lowest priorities to the results of this preprocessing step. However, adding an image space method narrows the possible speedup of our approach. Our results show that for most objects (including the heavily shaded ones), the object space feature extraction step is sufficient for a visually pleasing rendering during the interaction step. Our image reconstruction scheme converges progressively within a few frames as soon as the scene interaction stops. Therefore, missing shading details are added quickly with low visual annoyance. However, because of the computational overhead, the maximum performance boost, with respect to the maximum guaranteed frame rate with a visually acceptable result, decreases by approximately 10 frames per second. Using only the saliency information (without the object space features) results either in a very bad reconstruction result or a low speedup because of the rough image approximation and the low resolution of the saliency frame.

We have also evaluated the possibilities of image warping [Hauswiesner et al. 2010] for reusing the saliency information from previous (fully resolved) frames. This approach has shown promising results for a limited scene interaction and few object disocclusions, thus, for scenes with a high frame-to-frame coherence. However, image warping and saliency computations are computationally more expensive than the pure feature frame generation in object space. This fact lowers the maximum reachable performance; however, a pure image space method is not as accurate as the proposed object space approach.

10 Conclusions and future work

This paper presents a novel method for integrating NPR features into a rendering environment as a quality hint for the required granularity during a ray-based rendering without frame-to-frame coherence requirements. We show that higher frame rates are achievable during a scene interaction without a severe loss of image quality. Our method outperforms the state-of-the-art implementation of adaptive rendering, for example, delivered with the OptiX SDK,

Table 1: *An overview over the average rendering times for each step and different objects using our approach on our test system (variance $< 1\%$). For* geometry, *we tested the Stanford Dragon, Buddah, and Bunny datasets, our piggy dataset and some simpler drinking glass meshes, as shown in the accompanying video with the ray-traced 'glass' material from Figure 1(c). For* volume, *we tested the 512^3 datasets,* MANIX *and* FEET, *as shown in Figure 1(b) and (d) with different transfer functions. The measured times refer to a computation within a 1440×900 viewport. Note that not all rays on feature lines have to be computed. To obtain high guaranteed frame rates that maintain a visual appealing result, low priority features might be omitted by our algorithm from Section 5. The average ray count also varies with different viewports.*

	geometry $[ms]$	**volume** $[ms]$	**av. ray count** $[\#rays]$
feature frame	2	3	-
rays on contours	11	12	34.970
rays on ridges	12	14	37.208
rays on silhouette	5	8	9.373
rays on sug. highlights	10	19	23.043
rays on sug. contours	6	11	18.793
rays on valleys	5	7	10.288
reconstruction	15	15	17.756
sum	66	89	151.431
ground truth	208	251	2.457.600

in terms of speed and quality. It also reflects perceptual features more accurately due to its use of object space feature extraction than comparable approaches in image space.

We have performed a quantitative evaluation of the perceptual features to determine their impact on the visual quality and to show which features are best suited for adaptive ray-based image generation. Our algorithm can therefore also be used to achieve guaranteed frame rates by sorting the image pixels according to the feature priorities. Our algorithm is mainly intended for highly complex ray-based calculations, such as volume rendering, and for systems that require that object rendering does not occupy the whole computation unit (e.g., additional GPU-based simulation and segmentation).

We plan to perform a larger user study with different ray-tracing materials and ray-casted volumetric objects. From such a work, we expect a qualitatively founded classification of salient object features to answer the question of which feature works best for a particular type of object by means of human perception. In this work, we show evidence that contours are the most valuable feature of an object in terms of mathematically estimated image error and quality. However, to better qualify the remaining, less distinctive features, a deeper analysis will have to be performed with a sufficient number of human subjects.

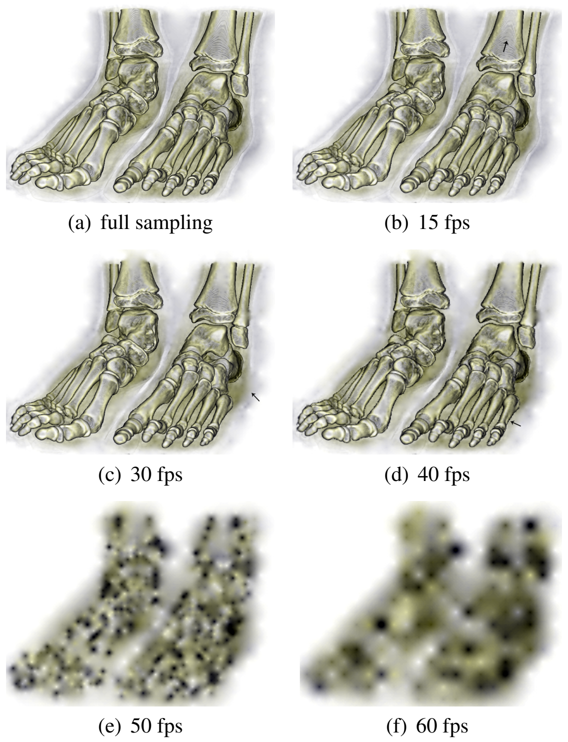
(a) full sampling (b) 15 fps (c) 30 fps (d) 40 fps (e) 50 fps (f) 60 fps

Figure 8: *This figure illustrates the decreasing quality with increasing requested guaranteed frame rates for a 512^3 volumetric dataset in a 1024×768 viewport. Image (b) still uses all available features to steer the sampling pattern and interpolates only more coarsely in feature-poor areas. This is visible in homogenous regions (e.g., at the small arrow in (b)). In (c) nearly all ridges and valleys and some parts of the silhouette are discarded for the interpolation, which can be seen for example at the left heel (at the small arrow in (c)). (d) leaves out also some rays which would be formed by contours, which causes coarser borders of the bones (e.g., at the small arrow in (d)). The interpolation in (e) and (f) has to discard all features, which results in the shown images.*

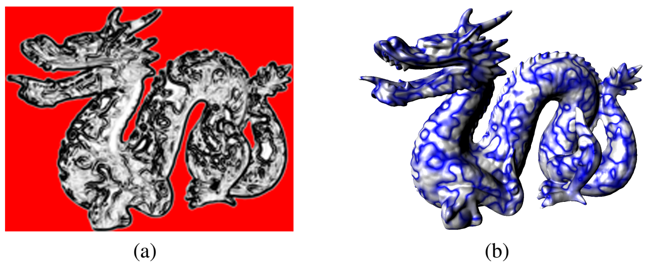
(a) (b)

Figure 9: *The visual saliency information can be included into our feature buffer (a) for the reconstruction of a ray-traced textured object (b). The saliency information was computed on a six-fold-smaller ground truth, the processed rays were reused for the final reconstruction. Colors in (a) have been encoded according to our definitions from Section 4.1.*

Acknowledgements

We would like to thank Morgan McGuire (Williams College) and Denis Kalkofen and Marc Streit (Graz University of Technology) for their valuable comments and Doug DeCarlo and Michael Burns (Princeton University) for providing their source code. This work was funded by the European Union in FP7 VPH initiative under contract number 223877 (IMPPACT) and the Austrian Science Fund (FWF) under contract P23329-N23.

References

AMIDROR, I. 2002. Scattered data interpolation methods for electronic imaging systems: a survey. *J. Electronic Imaging 11*, 2, 157–76.

BOADA, I., NAVAZO, I., AND SCOPIGNO, R. 2001. Multiresolution volume visualization with a texture-based octree. *The Visual Computer 17*, 185–197.

BRUCKNER, S. 2008. *Interactive Illustrative Volume Visualization*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

BURNS, M., KLAWE, J., RUSINKIEWICZ, S., FINKELSTEIN, A., AND DECARLO, D. 2005. Line drawings from volume data. *ACM Trans. Graph. 24*, 3, 512–518.

COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H. S., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2008. Where do people draw lines? *ACM Trans. Graph. 27*.

DANSKIN, J., AND HANRAHAN, P. 1992. Fast algorithms for volume ray tracing. In *VVS '92*, 91–98.

DAYAL, A., WOOLLEY, C., WATSON, B., AND LUEBKE, D. 2005. Adaptive frameless rendering. In *ACM SIGGRAPH 2005 Courses*, 24.

DECARLO, D., AND RUSINKIEWICZ, S. 2007. Highlight lines for conveying shape. In *NPAR'07*, 63–70.

DIPPÉ, M. A. Z., AND WOLD, E. H. 1985. Antialiasing through stochastic sampling. *SIGGRAPH CG 19*, 69–78.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01*, 9–16.

FERNANDO, R. 2004. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education.

FREUND, J., AND SLOAN, K. 1997. Accelerated volume rendering using homogeneous region encoding. In *VIS '97*, 191–ff.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The lumigraph. In *Computer graphics and interactive techniques*, SIGGRAPH'96, 43–54.

GROSSMAN, J. P., AND DALLY, W. J. 1998. Point sample rendering. In *Rendering Techniques 98*, 181–192.

HAUSWIESNER, S., KALKOFEN, D., AND SCHMALSTIEG, D. 2010. Multi-frame rate volume rendering. In *EGPGV'10*.

ITTI, L., KOCH, C., AND NIEBUR, E. 1998. A model of saliency-based visual attention for rapid scene analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence 20*, 1254–1259.

JESCHKE, S., WIMMER, M., SCHUMANN, H., AND PURGATHOFER, W. 2005. Automatic impostor placement for guaranteed frame rates and low memory requirements. In *ACM SIGGRAPH I3D*, 103–110.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *VIS '01*, 255–262.

KRAUS, M., AND ERTL, T. 2001. Topology-Guided Downsampling. In *VG 2001*, Springer Computer Science, 223–234.

LA MAR, E. C., HAMANN, B., AND JOY, K. I. 1999. Multiresolution techniques for interactive texture-based volume visualization. In *VIS'99*.

LEE, M. E., REDNER, R. A., AND USELTON, S. P. 1985. Statistically optimized sampling for distributed ray tracing. *SIGGRAPH CG 19*, 61–68.

LEVOY, M. 1990. Volume rendering by adaptive refinement. *The Visual Computer 6*, 1, 2–7.

LI, W., MUELLER, K., AND KAUFMAN, A. 2003. Empty space skipping and occlusion clipping for texture-based volume rendering. In *VIS '03*, 42.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH CG 21*, 4, 163–169.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH '97*, 199–208.

MARROQUIM, R., KRAUS, M., AND CAVALCANTI, P. R. 2008. Special section: Point-based graphics: Efficient image reconstruction for point-based and line-based rendering. *SIGGRAPH CG 32*, 189–203.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. *SIGGRAPH CG 21*, 65–72.

NOTKIN, I., AND GOTSMAN, C. 1997. Parallel progressive ray-tracing. *SIGGRAPH CG 16*, 1, 43–55.

PAINTER, J., AND SLOAN, K. 1989. Antialiased ray tracing by adaptive progressive refinement. *SIGGRAPH CG 23*, 281–288.

PARKER, S., MARTIN, W., PIKE J. SLOAN, P., SHIRLEY, P., SMITS, B., AND HANSEN, C. 1999. Interactive ray tracing. In *Interactive 3D Graphics*, 119–126.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: surface elements as rendering primitives. In *Computer graphics and interactive techniques*, SIGGRAPH'00, 335–342.

POMI, A., AND SLUSALLEK, P. 2005. Interactive Ray Tracing for Virtual TV Studio Applications. *J. Virtual Reality and Broadcasting 2*, 1.

RAMASUBRAMANIAN, M., PATTANAIK, S. N., AND GREENBERG, D. P. 1999. A perceptually based physical error metric for realistic image synthesis. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '99, 73–82.

SCHROEDER, W., MARTIN, K. M., AND LORENSEN, W. E. 1998. *The visualization toolkit (2nd ed.): an object-oriented approach to 3D graphics*. Prentice-Hall, Inc.

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T., AND HANRAHAN, P. 2008. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph. 27*, 3, 1–15.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *SIGGRAPH '95*, 351–358.

WALD, I., BENTHIN, C., AND SLUSALLEK, P. 2002. OpenRT – A Flexible and Scalable Rendering Engine for Interactive 3D Graphics. Tech. rep., CG Group, Saarland University.

WALD, I., FRIEDRICH, H., MARMITT, G., SLUSALLEK, P., AND SEIDEL, H.-P. 2005. Faster isosurface ray tracing using implicit kd-trees. *IEEE Vis and CG 11*, 562 –572.

WALD, I., MARK, W. R., GÜNTHER, J., BOULOS, S., IZE, T., HUNT, W., PARKER, S. G., AND SHIRLEY, P. 2007. State of the art in ray tracing animated scenes. In *STAR Proceedings of EG 2007*, 89–116.

WANG, Z., AND BOVIK, A. C. 2009. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine 26*, 1, 98–117.

WANG, Q., AND JAJA, J. 2008. Interactive high-resolution isosurface ray casting on multicore processors. *IEEE Vis and CG 14*, 3, 603 –614.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Image Processing 13*, 4, 600–612.

WEILER, M., WESTERMANN, R., HANSEN, C., ZIMMERMAN, K., AND ERTL, T. 2000. Level-of-detail volume rendering via 3d textures. 7–13.