Ray Prioritization Using Stylization and Visual Saliency

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

Abstract

This paper presents a new method to control scene sampling in complex ray-based rendering environments. It proposes to constrain image sampling density with a combination of object features, which are known to be well perceived by the human visual system, and image space saliency, which captures effects that are not based on the object's geometry. The presented method uses Non-Photorealistic Rendering techniques for the object space feature evaluation and combines the image space saliency calculations with image warping to infer quality hints from previously generated frames. In order to map different feature types to sampling densities, we also present an evaluation of the object space and image space features' impact on the resulting image quality. In addition, we present an efficient, adaptively aligned fractal pattern that is used to reconstruct the image from sparse sampling data. Furthermore, this paper presents an algorithm which uses our method in order to guarantee a desired minimal frame rate. Our scheduling algorithm maximizes the utilization of each given time slice by rendering features in the order of visual importance values until a time constraint is reached. We demonstrate how our method can be used to boost or stabilize the rendering time in complex ray-based image generation consisting of geometric as well as volumetric data.

Keywords: ray-tracing, ray-casting, volume rendering, photorealistic rendering, visual saliency

1. Introduction

A common challenge of high-quality ray-based image generation is maintaining the scene 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 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 image 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 only. These approaches obviously require the final result as an input for the optimal result, which is impossible. Hence, image regions from previously rendered frames [1] or sparsely sampled regions [2] are used.

We investigate the key element of adaptive approaches, which is the determination of which elements of an object can be coarsened and which must be preserved. Figure 1 outlines the core idea of this work.

Much perceptually based research has been performed in this area by researchers from different communities (e.g., the Non-Photorealistic Rendering (NPR) community). However, these results have mostly been used for scene stylization and scene enhancement so far. We present a new sampling strategy for ray-based image synthesis, which uses information about the scene, which is known to support the comprehension of 3D shapes [3] and visually attractive images areas in general [4]. In this paper, these 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.

Figure 1: This figure illustrates the basic idea of our method. Rays do not have to be equally distributed over the scene in ray-based rendering environments to get a visually pleasing result. It is sufficient to trace rays only in areas that have been proven to convey the shape of an object (left) for a good approximation of the result (right).

Our implementation produces a feature buffer for every frame that is efficient enough to be used during the ray generation as a lookup table for the required ray density. We derive a

feature priority map from the feature buffer that consists of object space features like silhouettes, suggestive contours, ridges and valleys, combined with image space features like the visual bottom up saliency information from previously rendered frames. All of them affect the ray density differently. 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 efficient visualization of volumetric datasets, we also demonstrated a way to apply our method to geometric objects with highly complex materials in ray-traced scenes.

This is an extend version of [5], which newly introduces the use of an image-based visual saliency analysis to capture the impact of lightning effects which go beyond pure geometric features and presents a thoroughly evaluation of our technique using the HDR-VDP-2 [6] visual metric for visibility and quality predictions. 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 method to calculate the visual saliency information efficiently enough from previous frames, so that this scene information gets applicable for a perceptually guided ray setup (Section 4.1).
- A progressively refineable sampling pattern, which is used to reconstruct sparsely sampled regions of the image (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 and a comparison to image space saliency information from previous frames (Section 6).

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 previous attempts at guaranteeing minimal frame rate are outlined in Section 2.4.

2.1. Interactive ray-based rendering

In this section, we briefly summarize the most common approaches to speed-up ray-based rendering algorithms. These approaches can roughly be divided into algorithmic improvements and the exploitation of successively available graphics hardware features.

Adaptive progressive rendering. Adaptive approaches, such as the one presented in this paper, aim 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 [7] and still used in many interactive ray based rendering systems [8, 9], 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 [10, 2, 11]. 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 [12]. 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 [13] depending on the required rendering time or used texture-based level of detail [14], topology guided downsampling [15] or multiple resolutions of the same dataset [16, 17].

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 [18, 19], most recent GPU approaches reach remarkable frame rates for low- to medium-complexity scenes for rendering in full quality [20, 21]. However, screen filling scenes or scenes with very high complexity are still too slow in order to meet 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 [22], iso-surface ray-casting [23, 24], ray pre-integration [25], homogeneous region encoding [26] and many kinds of direct GPU implementations [27, Chapter 39].

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 large uniform areas do [28, 2], to gain a visually acceptable result. Later systems adapt this assumption

towards the limitations of the human visual system. Ramasubramanian and colleagues [29] 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 with only 5-10% of the rays which have been used for a reference solution. Their results have visually only little to no difference to a fully computed ground truth.

Another popular method to extract visually attracting image areas is the calculation of the visual bottom-up saliency [4]. The saliency of an image is usually defined as a measure of how much a particular location contrasts with its surroundings, using dimensions such as color, orientation, motion, and depth. Hence, this method is especially suitable for analyzing cluttered scenes which are more dominated by texture rather than sharp edges [30, 31].

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 and colleagues [3], for example, show that object contours including the object silhouette are also used by artists to outline scenes. Several visualization algorithms and perceptually motivated work demonstrate that these features can be used to simplify complex scenes for a better understanding of the essential parts [32, 33, 34].

The features of an object or of a scene can be extracted in various ways: meshes can be analyzed in object space or, after rendering, their projection can be analyzed in image space. Similar methods apply 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 and the visual saliency information of a rendered object that we evaluate for ray decimation in this work. Our definitions of object features are similar to those from [34] and the definition of visual saliency is similar to that from [4].

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 [35].

Point-based rendering. Specialized approaches for computer graphics can be found for point-based rendering. The widely used *pull-push algorithm* [36] utilizes a pyramid algorithm for surface reconstruction. It has been adapted for the image-space reconstruction of under-sampled point-based surfaces [37]. Pfister and colleagues [38] extended this approach to fill the holes between splats. Unfortunately, these approaches are not suited for direct GPU implementations, as stated by Marroquim and colleagues [39]. The approach of Marroquim *et al.* [39], 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.

Image warping. Image warping [40] is a form of image-based rendering that allows to extrapolate new views from existing images with per-pixel depth information. Such methods can be useful if the changes between frames are small (i.e., during interaction with high frame-to-frame coherence), so that a new view can be reconstructed from previous frames. Artifacts occurring because of occlusions and disocclusions can be solved by a recalculation of those areas [41, 42].

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 colleagues [43] 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, [44] replaces complex objects optimally by impostors. These examples show that several applications require guaranteed frame rates but also that this problem is not well researched yet.

3. Overview over the method

Our approach consists of two passes. First, an importance buffer is created from object space features and image space features with the goal of encoding the visual importance of every image region for the perceivable image quality. The object space features are deduced from a set of line features which are extracted from the data set's corresponding meshes, projected to screen space, and classified according to our evaluation (see Section 6). The image space features are deduced by performing a saliency analysis of the previously rendered frame, warping the saliency to the current frame, and analyzing the result for disocclusions. The combined 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.

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

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

Figure 2: The happy Buddha object (a) rendered with different object space sparse line features (b-f) and an image space feature detector (g). Silhouettes (b), suggestive contours (c), suggestive highlights (d), ridges (e), valleys (f) and bottom-up visual saliency (g) are evaluated for ray decimation in this work. In the saliency image (g) dark areas correspond to highly salient regions, while white areas stand for zero saliency.

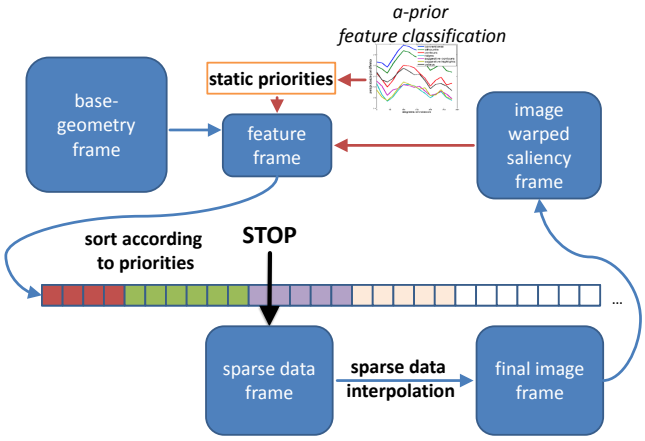


Figure 3: Overview of our prioritized rendering algorithm. Every ray's priority is computed according to Section 4.2 and is used to sort the pixels in a one-dimensional priority queue. Additionally, the visual saliency information can be image warped from the last frames as described in Section 4.1. The selection of features depends on the kind of scene and can therefore be modified by the user. The features are sorted in a priority queue and if a certain time limit is reached, the rendering process stops. The resulting sparse data frame can be reconstructed with the method from Section 4.4.

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 object space 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, we render an importance buffer in each frame (Section 4.1). This importance buffer is filled by a projection of object space line features to screen space combined with the result of an image space saliency analysis from the previously rendered frame. To yield the priority of every ray (Section 4.2), the entries from the importance buffer are combined with priority values from a fractal sampling pattern (Section 4.3).

4.1. Importance Buffer

The aim of the importance buffer is to estimate the importance of an image region for the overall perceivable image quality. A high importance value (close to 1.0) means that this area is important for the viewer to be able to understand the rendered objects. At the same time these areas will also strongly contribute to image quality. A low importance value (close to 0.0) means that this image area does not hold visually interesting features and that this region is rather homogeneous in color and intensity. We furthermore encode information about which areas correspond to pure background in the importance map to omit these areas from ray-tracing. To generate the importance buffer, we rely on object space features, which are exact and efficient to compute, as well as image space features, which can cover additional effects due to lighting or object textures. As there is a partial overlap between the two methods, we combine the feature-sets applying a per pixel max operator if both kinds of features are used at the same time. To benefit from our method, the importance buffer must be produced in a time frame that is shorter than the savings from the main rendering pass. Note that it is in general sufficient to compute the importance buffer at a lower resolution.

Object Space Features. To provide a sufficiently high frame rate in the first render pass, we have extended the approach from DeCarlo and colleagues [34] with selective GPU acceleration techniques, which are described in Section 7. 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. 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 4.3. The last option provides the highest flexibility. To remove any hidden features, we also render the underlying base mesh as fully opaque, homogeneous surface.

Image Space Features. Because pure object space features do not incorporate high frequencies in textures or lighting effects like shadows, reflections, or refractions, we also evaluate image space features. To detect these features in image space, we rely on bottom-up visual saliency in terms of color and intensity oppositions [4]. As we require an image to calculate the saliency, we face a chicken-egg problem. To determine the importance of an image region, we require the image. To efficiently create the image, we need to know where important areas are. Targeting interactive systems, we can make use of the previously generated view. We calculate a saliency buffer for the previous frame building an image pyramid using GLSL shaders. Afterwards, the saliency buffer is image warped according to the transformations that have been applied to the scene since the last frame was generated. Finally, small holes in the map are closed and big holes, probably due to disocclusions, are filled with a medium importance value. Warping the saliency buffer is less error prone than warping the image itself, because we only use it as basis for the decisions of which rays to cast into the scene and only warp from one frame to the other.

4.2. Adaptive subsampling

When the importance buffer 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 importance buffer contains a negative value 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. Otherwise, we consult an adjustable ray priority table, which defines an image space sampling pattern (see Section 4.3). Combining the importance buffer entry p_{imp} with the pattern priority p_{patt}, which is read from the sampling priority table, we can determine a per ray priority p_{ray}. The way the two independent priorities p_{imp} and p_{patt} are combined essentially captures the contribution to the image quality that can be expected when casting a certain number of rays in a region of a certain importance. To make this important quantity adjustable, we support an arbitrary mapping function f_{map} to combine these two priorities:

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

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

$$p_{ray} = \sum_{i+j\leq 2} \alpha_{i,j} \cdot p^i_{imp} \cdot p^j_{patt} \tag{2}$$

Rays are traced according to their priority p_{ray}. As the mapping function captures every ray's contribution to image quality, a fixed threshold can be used to trace rays with a high contribution only. Another option is to sort rays according to p_{ray} 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 importance buffer on the output image. Low weights for terms dominated by p_{imp} will result in a nearly uniform pattern, while a high contribution of p_{imp} creates samples at the feature areas only. 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. Homogeneous areas would contain only a few sampling points (see also Figure 8). All non-background rays, which have not been traced, are subject to a reconstruction step as described in Section 4.4.

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_{imp} (increasing $\alpha_{1,0}$ and $\alpha_{2,0}$), as most variation in color appears along the feature regions.

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). To define the sampling pattern, we start with a square pattern and a power of two, which defines the maximal distance between rays. In practice we place a ray every 8×8 pixel. In the next step, we place a sample at the center of the square which splits every square into four triangles. Together with the surrounding squares, which are also 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. The pattern can be refined locally and thus increase the sampling density for arbitrarily sized regions.

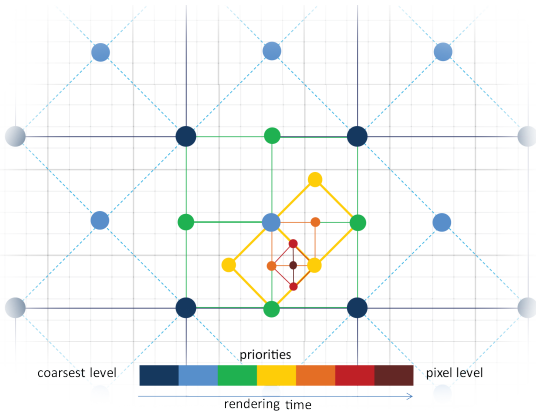


Figure 4: We use a sampling priority pattern similar to that outlined in this figure. For illustration reasons, this figure shows a much finer grid than would be used in reality. Blue defines the initial rays with priorities 1.0. With every sampling step, the pattern becomes finer, and the priority decreases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

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 [35]. 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, not enough lower density rays will be available to construct the missing points for the next higher density. For maximum performance, we make sure that the importance buffer is 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.

There are scenarios in which the aforementioned approach will fail. An unexpected load on the GPU, complex objects popping up in the scene, or highly different viewing positions prohibit 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 selected volume-rendered objects with different transfer functions and ray traced objects with and without textures and different physical properties (i.e., different reflection and refraction coefficients). Rendering at a lower resolu-

tion with bilinear interpolation served as a base-line condition. 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 the number of rays cast into the scene is increased. Rays are subsequently added according to their importance defined by tested feature evaluation strategy.

To quantify the visual improvements of different features, we use the currently most advanced image comparison metrics, the recently published HDR-VDP-2 [6] method. Image comparison means for our application to compare an image, which has been produced by our method, to the fully sampled ground truth image. HDR-VDP-2 is a very useful tool for predicting the visual quality as it is perceived by a human observer. The method provides two different comparison metrics.

The first metric is the *visibility* metric P_{map}, which gives an image in which each pixel represents the probability of detecting a difference between the two input images. To derive a single quantitative difference measure from this map, we compute its average value. This value can be seen as the average probability of detecting a difference between the fully traced ground truth image and the coarsely traced image. The second metric estimates a quasi-subjective *mean-opinion-score quality* predictor Q_{MOS}, which would normally require an extensive user-study. Therefore, the authors of HDR-VDP-2 have tested over 20 different variations of value pooling strategies and compared their predictions to two different image quality databases. HDR-VDP-2 revealed to be the best visibility and quality predictor [6] at the time of this work.

Figure 5 shows that all feature classifiers improve the image quality during interaction, compared with the conventional regular subsampling rendering. We believe the region between 5% and 100% of cast rays is the most interesting for our approach. It yields acceptable image qualities according to the mean-opinion-score quality measure Q_{MOS}. In this region, the average probability of detecting a difference between the full quality image and the image rendered with a reduced number of rays P_{map} is approximately twice as high for the conventional regular subsampling approach than for any of the proposed feature detection methods. We observed that the image-based feature detector based on visual saliency performs very well for low ray counts, while its performance is approximately equal to the object space detectors for medium to high ray counts. In terms of object space feature detectors, using a combination of exterior silhouettes and contours works best. Using the exterior silhouette leads to the best result for objects with a low interior feature count (e.g., glass objects). For volumetric objects, the exterior silhouette initially does not necessary improve the image quality, because it might cover the whole dataset and exclude the (probably more important) internal features. Ridges and valleys as well as suggestive contours and highlights overall contribute less to image quality than contours or silhouettes. If strongly textured objects are rendered, object space line features cannot predict the visual outcome, which results in a performance comparable to conventional regular subsampling. However, applying our image space visual saliency measure significantly increases the image quality.

Based on our observations from automatic tests, we first

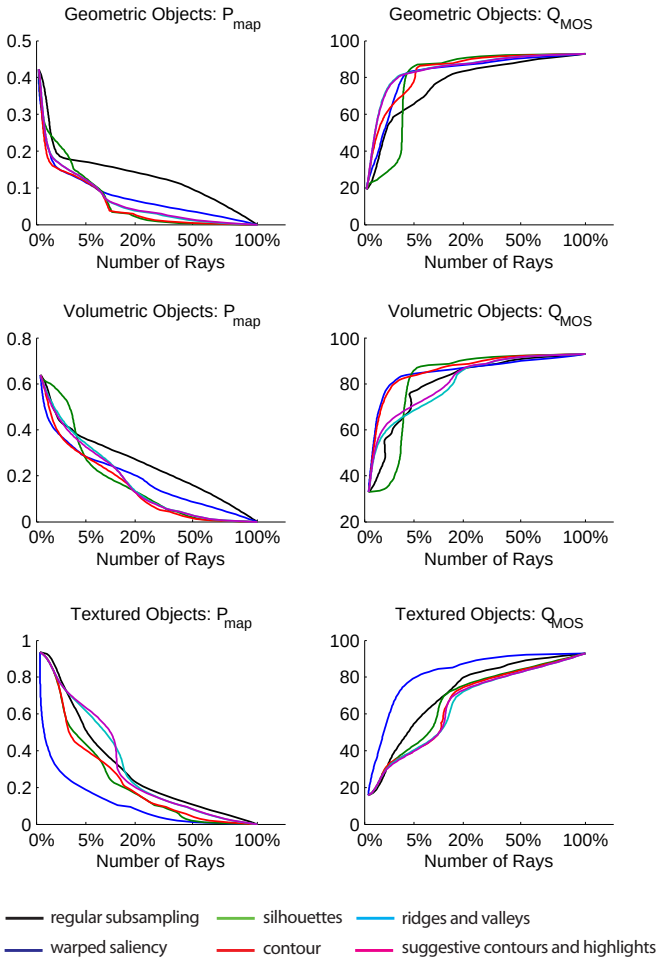


Figure 5: These graphs compare the render quality in terms of the visibility (P_{map}) and mean-opinion-score quality (Q_{MOS}) reached by tracing a certain number of rays according to different feature classifiers. The left column shows the result for the difference predictor P_{map} and the right columns shows the results obtained for the mean-opinion-score predictor Q_{MOS}. Each row represent a different use case scenario, whereas the numerical results were obtained from averaging multiple data sets of varied complexity. For visual examples of the different classifiers, please see figure 2.

roughly classify the object space line features into *strong visual features* and *medium visual features*. We do 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. *Exterior silhouettes* also yielded a good result in all of the tested scenes. 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 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 option have shown that users tend to fully disable object space features such as suggestive contours, suggestive highlights, ridges and valleys to gain higher frame rates. These features have also shown a low visual improvement during our offline evaluation as previously outlined. Most users preferred to disable the image space feature generation based on visual saliency for untextured objects and volumetric datasets. For textured objects, most users exclusively used the image space visual saliency warping for the importance buffer generation.

7. Implementation

In this section, we describe the implementation details of our rendering pipeline as shown in Figure 6. The preprocessing and feature generation stages use a combination of OpenGL and CUDA. We have implemented the main part of our method as part of the OptiX SDK [21], 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. Mesh extraction

The object space feature evaluation requires a smooth surface mesh. 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, an input mesh is directly available for the mesh preparation 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 [45]). 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

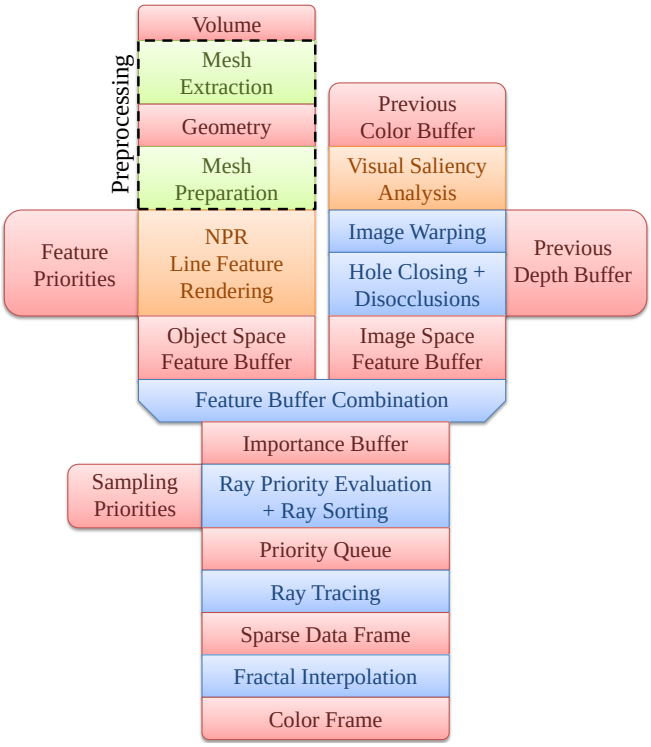


Figure 6: The rendering pipeline used for our perceptually guided ray tracing. Red boxes denote data objects, green boxes represent CUDA kernels, orange boxes indicate shader based OpenGL rendering, and blue boxes represent Optix Kernel calls.

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 marching cubes [46] algorithm) to extract the iso-surfaces whenever the transfer function is changed. Because the resulting meshes are over-tessellated, we simplify this mesh with a GPU-based simplification method as described in the following paragraphs.

7.2. 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 [47] and mesh simplification [48] 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.

7.3. NPR Line Feature Rendering

The object space feature frame is rendered by using an extended version of the publicly available framework provided by [34], 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. normal vector $\times$ viewing vector per geometry vertex) 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.

For volume rendering, we have also attempted a direct feature line extraction as proposed by Burns et al. [33]. Experiments with the Burns system have shown that the frame rates are not as high as those obtained with iso-surfaces, which we use with 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 to 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 for certain cases.

7.4. Visual Saliency Analysis

To generate the image space feature frame, we require a previously rendered frame. In order to match the current view as closely as possible, we use the last rendered frame and compute its bottom-up visual saliency [4]. We use *GLSL shaders* to convert the image into the CIE_{Lab} space and to build the image pyramid required for the saliency computation. To efficiently build the image pyramid, we applying separable Gaussian filtering with multiple render targets, each representing a different scale. In a final pass, the saliency of each pixel is computed from the image pyramid.

7.5. Image Warping and Hole Closing

The saliency buffer is mapped as a texture in the OptiX context. In a first launch, we use the depth information from the previously generated frame to warp the saliency buffer to match the current view. Because the previously generated frame was also constructed using our adaptive method, its depth information may be available as a sparsely populated buffer only. Hence, holes may arise during the warping process. In a second pass, we close these holes using a nearest neighbor search with a search frame size chosen according to the sampling pattern of the last frame. If no warped information is found in the vicinity of the target position, the hole most likely stems from a disocclusion. We fill these holes with a medium importance value of 0.5. This empirically determined value proved to be a good compromise between ignoring unknown previously hidden areas and rendering them in full quality.

7.6. Importance Buffer and Ray Setup

The importance buffer is formed from the feature buffers using a max operator. The ray priority is determined from the importance buffer and a fixed sampling priority as described in Section 4.2. After the rays have been sorted according to their priority the ray tracing is carried out. To reduce the number of ray fetches from the priority queue, rays are fetched in groups of 32. For the ray tracing itself we use a *pinhole camera* model.

7.7. 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 [21] 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.

7.8. Fractal Interpolation

The reconstruction based on the fractal interpolation scheme as described in Section 4.3 requires two passes. In the first pass, we reconstruct those anchor points, which have four available surrounding anchor points. This pass guarantees smooth transitions from one fractal level to the next. In the second pass, we use information form the importance buffer and the priority queue fill level to estimate the trace level in the vicinity of the pixel. Based on this initial guess, we search for available anchor points and interpolate the pixel's color from the found anchor points.

8. Results

In Section 6, we present our results on how much a certain 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 7 shows the increase of the *visibility* metric P_{map} and the decrease of the *mean-opinion-score quality* predictor Q_{MOS} for increasing guaranteed frame rates. With our feature adaptive sampling strategy, non important rays are omitted first, which increases framerates dramatically while the image quality is influenced little.

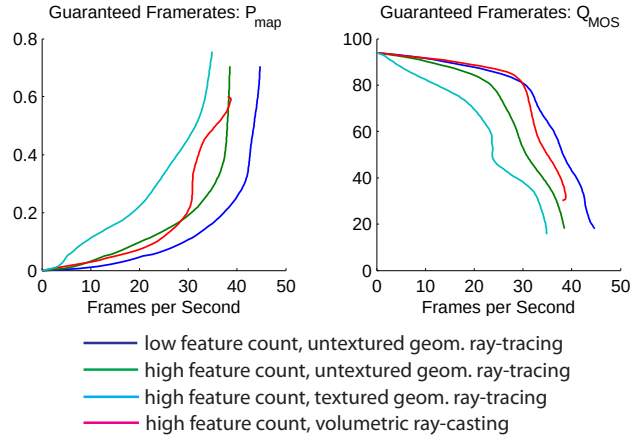


Figure 7: These plots show the increase of the visibility metric P_{map} and the decrease of the mean-opinion-score quality predictor Q_{MOS} for increasing guaranteed frame rates averaged over different data sets. The viewport for this test was 1024×768. Note that, after a certain guaranteed frame rate, the time frame becomes too short to render the important features. This issue becomes apparent in the plots by the bend between 25 and 40 fps.

The feature importance buffer can be generated 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. 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 computation times of our method compared with the unaltered ground truth. Figures 8, 9 and 10 show the decrease of quality with an increasing frame rate demand for different object types. The image quality remains subjectively stable, as long as the frame rate is reasonably adjusted. Figure 8 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 9 shows the quality decrease for a textured object and the associated image space feature frames. Figure 10 shows the quality decrease for a volumetric object.

9. Conclusions and future work

This paper presents a novel method for integrating perceptual features into a rendering environment as a quality hint for the required granularity during a ray-based rendering. To analyze the scene for geometric features, we utilize Non-Photorealistic Rendering techniques, which can be evaluated efficiently. To analyze textured objects, we utilize image space saliency combined with image warping to infer information from previously rendered frames. We show that higher frame rates are achievable during the 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, in terms of speed and quality.

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. For *volume*, we tested the 512^3 datasets, *MANIX* and *FEET*, with different transfer functions. The measured times refer to a computation within a 1440×900 viewport. The object space feature frame and the saliency warping was performed in full resolution, which is in general not necessary. 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.

	geometry [*ms*]	**volume** [*ms*]	**av. ray count** [#*rays*]
object space features	2	3	-
saliency warping	6	6	-
rays on contours	10	11	31.461
rays on ridges	8	9	23.356
rays on silhouette	5	7	9.251
rays on sug. highlights	8	9	17.122
rays on sug. contours	5	7	15.549
rays on valleys	5	6	9.646
rays on saliency	12	13	35.678
reconstruction	15	15	1.153.937
sum	76	86	1.296.000
ground truth	208	251	1.296.000

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., concurrently performed 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, even though the HDR-VDP-2 metric approximates the outcome of a prospective user-study about the visual quality of images already quite well, as shown by Mantiuk et al. [6].

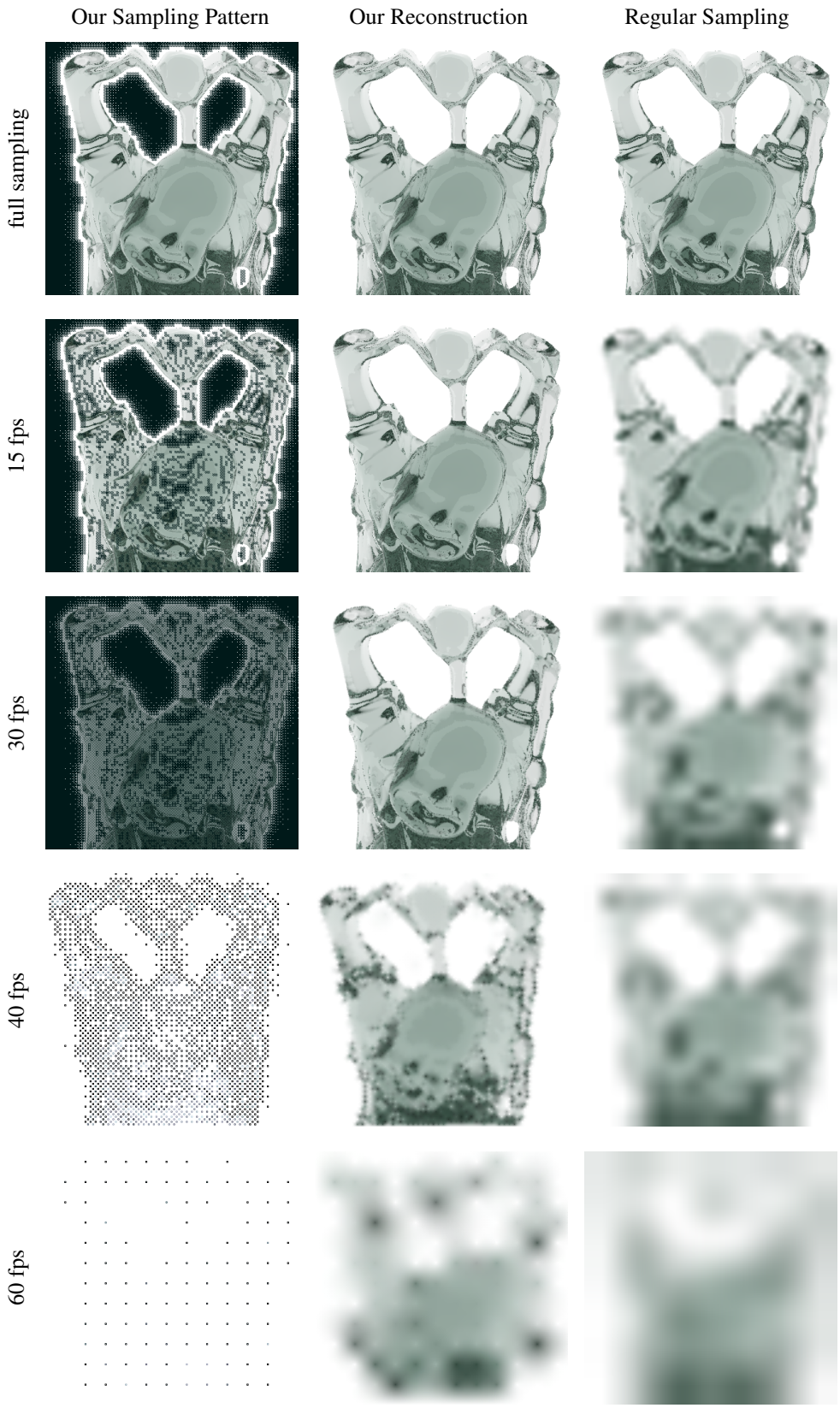
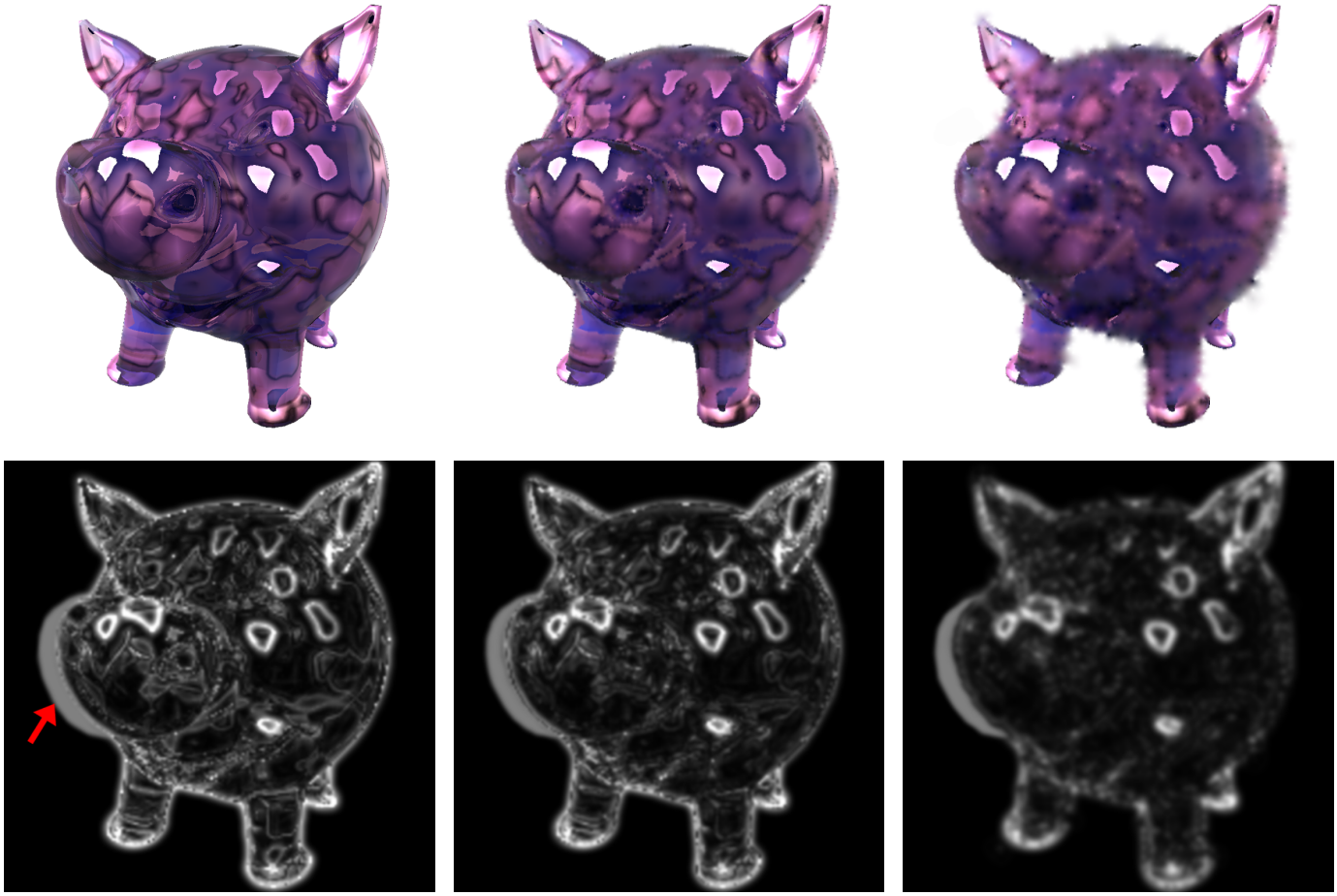
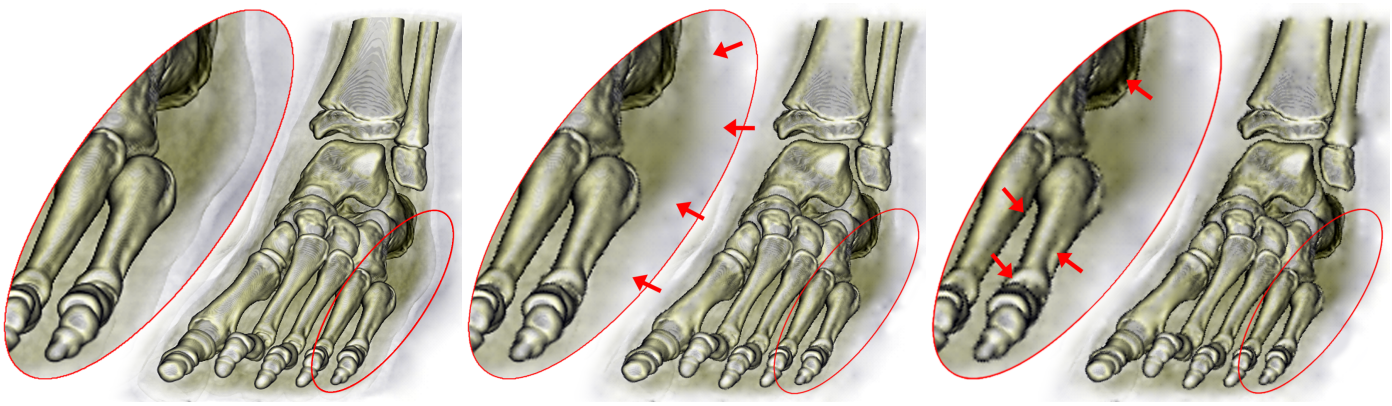


Figure 8: This figure illustrates the decreasing ray count with increasing requested guaranteed frame rates for the Happy Buddha dataset in a 1024×768 viewport. The left column shows the actual pixels that have been traced, and the middle column shows the result with our fractal pattern interpolation scheme. The 40 fps and 60 fps images of the left column are the edge images of the traced pixels to emphasize their positions in the printed versions of this paper. The right column shows the results using a regular sub-sampling pattern with a linear interpolation for comparison.



(a) full sampling (b) 20 fps (c) 30 fps

Figure 9: This figure illustrates the decreasing quality with increasing requested guaranteed frame rates for the Piggy data set in a 1024×768 viewport rendered with a complex lighting model combining procedural texturing, reflections, and refractions. The textural effects cannot be detected by the geometry-based object space feature detectors, thus we also apply an image space saliency detector. The second row shows the response of this saliency detector applied to a four-fold-smaller version of the previously rendered frame and warped according to the current view. The gray area next to the pig's mouth (red arrow) indicates a disocclusion, for which no information from the previous frame is available due to viewpoint motion. Note that the saliency response is especially high at the borders of the reflective patches. One problem of the image space saliency evaluation is that by decreasing the number of traced rays the image gets more blurry and thus the saliency response is also less accurate.



(a) 5 fps (full sampling) (b) 30 fps (c) 40 fps

Figure 10: This figure illustrates the decreasing quality with increasing requested guaranteed frame rates for a 512^3 volumetric dataset in a 1024×768 viewport. In image (b) the low priority features at the side are discarded to reach the requested frame rate. The important parts of the foot are still very well presented. To meet 40 fps (c), the number of contour forming rays is reduced, which causes coarser borders of the bones.

Acknowledgements

We would like to thank Morgan McGuire (Williams College) 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

[1] A. Dayal, C. Woolley, B. Watson, D. Luebke, Adaptive frameless rendering, in: ACM SIGGRAPH 2005 Courses, SIGGRAPH'05, 2005, p. 24.

[2] J. Painter, K. Sloan, Antialiased ray tracing by adaptive progressive refinement, ACM SIGGRAPH Computer Graphics, SIGGRAPH '89 Proceedings of the 16th annual conference on Computer graphics and interactive techniques 23 (3) (1989) 281–288.

[3] F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein, T. Funkhouser, S. Rusinkiewicz, Where do people draw lines?, ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008 27 (3) (2008) 88:1–88:11.

[4] L. Itti, C. Koch, E. Niebur, A model of saliency-based visual attention for rapid scene analysis, IEEE Trans. on Pattern Analysis and Machine Intelligence 20 (11) (1998) 1254–1259.

[5] B. Kainz, M. Steinberger, S. Hauswiesner, R. Khlebnikov, D. Schmalstieg, Stylization-based ray prioritization for guaranteed frame rates, in: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering, NPAR '11, ACM, 2011, pp. 43–54.

[6] R. Mantiuk, K. J. Kim, A. G. Rempel, W. Heidrich, HDR-VDP-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions, ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2011 30 (11) (2011) 40:1–40:14.

[7] D. P. Mitchell, Generating antialiased images at low sampling densities, ACM SIGGRAPH Computer Graphics , SIGGRAPH '87 Proc. 14th annual conference on Computer graphics and interactive techniques 21 (4) (1987) 65–72.

[8] W. Schroeder, K. M. Martin, W. E. Lorensen, The visualization toolkit (2nd ed.): an object-oriented approach to 3D graphics, Prentice-Hall, Inc., 1998.

[9] I. Wald, C. Benthin, P. Slusallek, OpenRT – A Flexible and Scalable Rendering Engine for Interactive 3D Graphics, Tech. rep., CG Group, Saarland University (2002).

[10] M. A. Z. Dippé, E. H. Wold, Antialiasing through stochastic sampling, ACM SIGGRAPH Computer Graphics, SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques 19 (3) (1985) 69–78.

[11] I. Notkin, C. Gotsman, Parallel progressive ray-tracing, Computer Graphics Forum 16 (1) (1997) 43–55.

[12] M. Levoy, Volume rendering by adaptive refinement, The Visual Computer 6 (1) (1990) 2–7.

[13] J. Danskin, P. Hanrahan, Fast algorithms for volume ray tracing, in: Proc. 1992 workshop on Volume visualization, VVS '92, 1992, pp. 91–98.

[14] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, T. Ertl, Level-Of-Detail volume rendering via 3d textures, in: Proc. 2000 IEEE symposium on Volume visualization, VVS'00, 2000, pp. 7–13.

[15] M. Kraus, T. Ertl, Topology-Guided Downsampling, in: Proc. Volume Graphics 2011, Springer Computer Science VG'11, Springer Verlag, Wien, New York, 2001, pp. 223–234.

[16] E. C. La Mar, B. Hamann, K. I. Joy, Multiresolution techniques for interactive texture-based volume visualization, in: Proc. 10th IEEE Visualization 1999, VIS'99, IEEE Computer Society, 1999, pp. 355–361.

[17] I. Boada, I. Navazo, R. Scopigno, Multiresolution volume visualization with a texture-based octree, The Visual Computer 17 (3) (2001) 185–197.

[18] S. Parker, W. Martin, P. Sloan, P. Shirley, B. Smits, C. Hansen, Interactive Ray Tracing, in: Proc. Interactive 3D Graphics, I3D'99, 1999, pp. 119–126.

[19] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, P. Shirley, State of the art in ray tracing animated scenes, in: STAR Proceedings of EG 2007, 2007, pp. 89–116.

[20] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, P. Hanrahan, Larrabee: a many-core x86 architecture for visual computing, ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008 27 (3) (2008) 18:1–18:15.

[21] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, M. Stich, Optix: a general purpose ray tracing engine, ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2010 29 (4) (2010) 66:1–66:13.

[22] W. Li, K. Mueller, A. Kaufman, Empty space skipping and occlusion clipping for texture-based volume rendering, in: Proc.14th IEEE Visualization 2003, VIS'03, 2003, p. 42.

[23] I. Wald, H. Friedrich, G. Marmitt, P. Slusallek, H.-P. Seidel, Faster isosurface ray tracing using implicit KD-trees, IEEE Trans. on Visualization and Computer Graphics 11 (5) (2005) 562–572.

[24] Q. Wang, J. JaJa, Interactive high-resolution isosurface ray casting on multicore processors, IEEE Trans. on Visualization and Computer Graphics 14 (3) (2008) 603–614.

[25] K. Engel, M. Kraus, T. Ertl, High-quality pre-integrated volume rendering using hardware-accelerated pixel shading, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, HWWS'01, 2001, pp. 9–16.

[26] J. Freund, K. Sloan, Accelerated volume rendering using homogeneous region encoding, in: Proceedings of the 8th conference on Visualization '97, VIS'97, 1997, pp. 191–ff.

[27] R. Fernando, GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics, Pearson Higher Education, 2004.

[28] M. E. Lee, R. A. Redner, S. P. Uselton, Statistically optimized sampling for distributed ray tracing, Proc. 12th annual conference on Computer graphics and interactive techniques 19 (3) (1985) 61–68.

[29] M. Ramasubramanian, S. N. Pattanaik, D. P. Greenberg, A perceptually based physical error metric for realistic image synthesis, in: Proc. 26th annual conference on Computer graphics and interactive techniques, SIGGRAPH '99, ACM, 1999, pp. 73–82.

[30] D. Gao, N. Vasconcelos, Discriminant Saliency for Visual Recognition from Cluttered Scenes, in: L. K. Saul, Y. Weiss, l. Bottou (Eds.), Advances in Neural Information Processing Systems, Vol. 17, MIT Press, 2005, pp. 481–488.

[31] D. Gao, V. Mahadevan, N. Vasconcelos, On the plausibility of the discriminant center-surround hypothesis for visual saliency., Journal of Vision 8 (7) (2008) 13:1–13:18.

[32] S. Bruckner, Interactive illustrative volume visualization, Phd-thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria (Mar. 2008).

[33] M. Burns, J. Klawe, S. Rusinkiewicz, A. Finkelstein, D. DeCarlo, Line drawings from volume data, ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 200 24 (3) (2005) 512–518.

[34] D. DeCarlo, S. Rusinkiewicz, Highlight lines for conveying shape, in: International Symposium on Non-Photorealistic Animation and Rendering (NPAR), NPAR'07, 2007, pp. 63–70.

[35] I. Amidror, Scattered data interpolation methods for electronic imaging systems: a survey, J. Electronic Imaging 11 (2) (2002) 157–176.

[36] S. J. Gortler, R. Grzeszczuk, R. Szeliski, M. F. Cohen, The lumigraph, in: Proc. 23rd annual conference on Computer graphics and interactive techniques, SIGGRAPH'96, 1996, pp. 43–54.

[37] J. P. Grossman, W. J. Dally, Point sample rendering, in: Rendering Techniques 98, Springer Berlin / Heidelberg, 1998, pp. 181–192.

[38] H. Pfister, M. Zwicker, J. van Baar, M. Gross, Surfels: surface elements as rendering primitives, in: Proc. Computer graphics and interactive techniques, SIGGRAPH'00, 2000, pp. 335–342.

[39] R. Marroquim, M. Kraus, P. R. Cavalcanti, Special section: Point-based graphics: Efficient image reconstruction for point-based and line-based rendering, Computers and Graphics 32 (2) (2008) 189–203.

[40] L. McMillan, G. Bishop, Plenoptic modeling: an image-based rendering system, in: Proc. 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH '95, ACM, 1995, pp. 39–46.

[41] W. R. Mark, L. McMillan, G. Bishop, Post-rendering 3d warping, in:

Proc. 1997 symposium on Interactive 3D graphics, I3D'97, ACM, 1997, pp. 7–16.

[42] S. Hauswiesner, D. Kalkofen, D. Schmalstieg, Multi-frame rate volume rendering, in: Eurographics Symposium on Parallel Graphics and Visualization, EGPGV'10, Eurographics Association, 2010, pp. 19–26.

[43] A. Pomi, P. Slusallek, Interactive Ray Tracing for Virtual TV Studio Applications, J. Virtual Reality and Broadcasting 2 (1) (2005) 1–10.

[44] S. Jeschke, M. Wimmer, H. Schumann, W. Purgathofer, Automatic impostor placement for guaranteed frame rates and low memory requirements, in: Proceedings of the 2005 symposium on Interactive 3D graphics and games, I3D'05, ACM, 2005, pp. 103–110.

[45] J. Kniss, G. Kindlmann, C. Hansen, Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets, in: Proceedings of the conference on Visualization '01, VIS '01, 2001, pp. 255–262.

[46] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, ACM SIGGRAPH Computer Graphics, SIGGRAPH '87 Proc. 14th annual conference on Computer graphics and interactive techniques 21 (4) (1987) 163–169.

[47] G. Taubin, A signal processing approach to fair surface design, in: Proc. 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH '95, 1995, pp. 351–358.

[48] D. Luebke, C. Erikson, View-dependent simplification of arbitrary polygonal environments, in: Proc. 24th annual conference on Computer graphics and interactive techniques, SIGGRAPH'97, 1997, pp. 199–208.