EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (2001), Number 3

Perceptually Guided Corrective Splatting

Jörg Haber, Karol Myszkowski, Hitoshi Yamauchi, and Hans-Peter Seidel

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract

One of the basic difficulties with interactive walkthroughs is the high quality rendering of object surfaces with non-diffuse light scattering characteristics. Since full ray tracing at interactive rates is usually impossible, we render a precomputed global illumination solution using graphics hardware and use remaining computational power to correct the appearance of non-diffuse objects on-the-fly. The question arises, how to obtain the best image quality as perceived by a human observer within a limited amount of time for each frame. We address this problem by enforcing corrective computation for those non-diffuse objects that are selected using a computational model of visual attention. We consider both the saliency- and task-driven selection of those objects and benefit from the fact that shading artifacts of "unattended" objects are likely to remain unnoticed. We use a hierarchical image-space sampling scheme to control ray tracing and splat the generated point samples. The resulting image converges progressively to a ray traced solution if the viewing parameters remain unchanged. Moreover, we use a sample cache to enhance visual appearance if the time budget for correction has been too low for some frame. We check the validity of the cached samples using a novel criterion suited for non-diffuse surfaces and reproject valid samples into the current view.

1. Introduction

Interactive navigation through a virtual environment is a powerful tool in many engineering and design applications, for instance in architecture and interior design, lighting design, and simulation. In general, these applications require global illumination solutions. In the near future, it can be expected that the importance of interactive exploration will increase significantly due to common applications such as virtual tourism, virtual museums, virtual shopping, and the like. Such applications will be running on powerful servers accessible through the Internet and will be capable of handling complex scenes whose appearance, realism, and believability would benefit greatly from global illumination solutions. Thus, there is a strong quest towards improving the performance of interactive rendering involving global illumination.

Many existing solutions are based on precomputing so-called illumination maps and their rendering using graphics hardware. Since the computation time at the preprocessing stage is not critical in many applications, illumination maps of high quality can be prepared for complex environments. A severe limitation of illumination maps is that basically only non-directional light components can be efficiently stored. Storing precomputed lighting for specular surfaces typically involves huge storage costs and/or degradation of image quality. As a consequence, proper appearance of directional lighting — resulting from light scattering by glossy, mirror-like, or transparent surfaces — can be achieved only by on-the-fly computations. Such computations must be repeated for each change of the viewing parameters during an interactive session.

A reasonable approximation of the appearance of non-diffuse surfaces can be obtained at interactive frame rates using graphics hardware functionalities such as Phong shading, environment mapping, and custom designed techniques. While the quality of the resulting images can be satisfactory for less demanding applications such as computer games, the fidelity of appearance of directional lighting effects with respect to their real world counterpart is poor. Clearly, some physically-plausible tools such as ray tracing are required for this purpose. However, this leads to a substantial increase of computation cost. Due to this cost, only a limited number of pixels can be updated for every frame without impairing the sense of interactivity for the user, where at least 5–10 fps are required. Thus, the question arises how to efficiently allot the system resources in order to minimize the perceptual distance between the images obtained during an interactive session and their corresponding ray traced images.

An important issue is the choice of image regions for the ray tracing computation, which should correspond to their ability of attracting visual attention. Visual attention is one of the fundamental characteristics of the human visual system. It is often compared to a spotlight, which enhances information within a selected region of the viewed image while suppressing information in the remaining regions [30]. It turns out that regions that are strong attractors of visual attention are highly correlated between subjects and usually affect a limited screen area [46, 22, 24]. Thus, ordering the ray tracing computations according to the predicted saliency of non-diffuse objects is a promising strategy, which should reduce the image artifacts as perceived by a human observer.

We present an approach for interactive navigation in photometrically complex environments with arbitrary light scattering characteristics. At the preprocessing stage we use the Density Estimation Particle Tracing technique [39]. This technique is capable of handling surfaces with arbitrary light scattering functions during global illumination computation. However, the final illumination maps are stored exclusively for the diffuse light component. To reduce the complexity of the mesh-based illumination maps, we perform mesh-to-texture postprocessing as described in [21]. During interactive rendering, graphics hardware is used to display the precomputed view-independent part of the global illumination solution using illumination maps. Objects with directional characteristics of scattered lighting are ray traced in the order corresponding to their saliency as predicted by the state-of-the-art visual attention model developed by Itti [17, 16]. Since the original model of Itti is purely saliency-driven, we extend this model to take into account volition-controlled and task-dependent attention, which is especially important for interactive applications. A hierarchical sampling technique is used to cover the image region representing an “attractive” object rapidly with point samples. These point samples are splatted into the frame buffer using a footprint size that depends on the hierarchy level of the samples. To ensure that this corrective splatting affects only those objects, for which the samples have been computed, we use the stencil test feature of the graphics hardware together with the stencil mask that has been created during rendering. Sample caching is performed to reuse samples computed for previous frames. A novel approach for aging samples is proposed, which ensures proper reconstruction of the appearance of specular and transparent objects. The correction algorithm operates in a progressive manner: the displayed image converges towards a high quality ray traced solution, if the viewing parameters do not change anymore. Rendering of illumination maps, corrective splatting, and evaluation of visual attention models are implemented as independent and asynchronously operating threads, which perform best on multi-processor operation platforms.

2. Previous Work

The discussion of previous work is focused on two main issues. First, we briefly review existing solutions for interactive rendering of global illumination solutions. In particular, we focus on the problem of high quality rendering of surfaces with directional light scattering characteristics. In the second part we discuss the perception-based guidance of such computations. Here, we are especially interested in the application of visual attention models within the context of interactive rendering, which mostly remains an unexplored problem so far.

2.1. Interactive Rendering of Global Illumination

In the last couple of years, progress in the development of fast (and sometimes parallel) systems with comparatively large main memory and specialized graphics hardware has led to increased research activity concerning interactive rendering of complex scenes. The complexity of a scene can be split into *geometric complexity* (i.e., the number of polygons or other geometric primitives that have to be processed to render a view of the scene) and *photometric complexity*. The latter is determined by the complexity of the materials/shaders, number and types of light sources, and the complexity of the rendering equation that is used to compute the global and local illumination.

Geometric complexity can be addressed by *level-of-detail (LOD)* techniques such as, e.g., progressive meshes [15] or impostors [19, 28, 31, 8]. In general, LOD techniques take into account geometric properties only and do not consider photometric properties.

A number of methods have been developed to compute high quality non-diffuse global illumination effects, see for instance [12] for a good overview. In general, these methods perform a large number of view-dependent computations tracing light paths through the scene. Thus, the resulting computation times prohibit interactive rendering. Although the computation times can be drastically reduced by decoupling view-independent (e.g., diffuse) and view-dependent (e.g., specular) global illumination effects using *multi-pass global illumination* techniques [40, 36, 18, 35], interactive frame rates are still hard to obtain.

Due to the performance and enhanced functionality of current graphics hardware, a number of high quality shading effects can be computed at interactive frame rates using *multi-pass OpenGL rendering* [29, 23, 6, 9, 14, 41, 1]. The effects that can be simulated include different kinds of shadows, specular and glossy reflections on curved objects, approximations of arbitrary reflection functions, bump maps, and normal maps. Each of these techniques handles some special case(s) from the broad range of lighting effects, and it is not always obvious how to combine them. Moreover, there are typically some constraints imposed on the scene, e.g., a limited number of light sources, restricted curvature of reflectors, minimum distance between objects and the reflected environment, and the like.

Another way to achieve interactive frame rates for high quality shading is to perform *massively parallel computations* [25], which can be coupled with a progressive display process [45] or with graphics hardware-assisted techniques [38]. The render cache [42] is a fully decoupled display process that caches and reprojects a set of samples. Outdated samples are replaced by new samples from ray tracing or path tracing clients using a priority-based error diffusion sampling strategy. In contrast to many other approaches, the render cache does not rely on 3D graphics hardware. During camera motion, however, black holes may appear between the reprojected samples despite of some smoothing filter, as long as no new samples have been generated by the ray tracing clients.

The concept of sparse sampling is exploited by discontinuity meshing [26], which may additionally be combined with caching of samples [32]. Rendering the resulting triangle mesh using hardware Gouraud shading yields a piecewise linear approximation to the final image. Another approach based on sparse sampling uses projective textures to correct the appearance of non-diffuse objects during OpenGL rendering of a precomputed radiosity solution [34]. Samples obtained from ray tracing clients are splatted into textures, which are resized adaptively according to the sample density.

2.2. Perception-Based Rendering

The main goal of perception-guided rendering techniques is to save computation without compromising the resulting image quality as perceived by a human observer. Mitchell [20] was the first to apply perceptual considerations to the design of an adaptive sampling pattern in ray tracing, which leads to a reduction of visible noise. Instead of using absolute differences in sample intensity, he considered contrast (i.e., ratio of intensities), which in terms of perception is a better measure of the variation between samples. Bolin and Meyer [2] developed a simple vision model to guide ray casting and were able to reduce the number of secondary rays for image regions with higher spatial frequencies. This was the first attempt to include visual masking [30] into a rendering algorithm. An even more sophisticated masking model suitable for rendering was developed by Ferwerda *et al.* [11]. Recently, some more advanced perception-based error metrics have been developed to guide adaptive sampling in order to reconstruct chromatic and achromatic spatial detail and to compensate for masking effects [3, 27]. While all discussed solutions have been designed specifically for rendering static images, their extension to interactive applications seems to be feasible. The main problem is the reduction of the computational overhead involved in perception modeling.

Since adding more complexity to current early vision models typically yields only small improvements [24], we investigate higher level perceptual and cognitive elements such as visual attention models to improve rendering performance. Yee [47] was the first to use a visual attention model to improve the efficiency of indirect lighting computations in the RADIANCE system [44] for dynamic environments. A saliency map with topographic locations of the attention attractors is created using the visual attention model developed by Itti [16]. The map is used to control the caching of indirect lighting on a per-pixel basis. For less salient image regions, greater errors can be tolerated and the indirect lighting can be interpolated for a larger neighborhood. This makes caching more efficient at the expense of blurring details in the reconstructed illumination solution.

Yee reported that by considering visual attention, a significant rendering speed-up can be achieved without noticeable degradation of animation quality. However, Yee noticed that subjects who had seen the same animation a number of times were able to notice more and more artifacts, because their attention was drifting to less and less salient image features. In our application, we deal with interactive, user-driven rendering rather than with fixed animation sequences and we believe that it is less likely to obtain similar sequences of images multiple times in row. Even if it would be the user's intention to do so, our sample caching scheme leads to better and better quality of images when similar views are revisited. Moreover, our goal is not to use the model of Itti to choose the final level of rendering accuracy for various image regions as in [47]. We rather predict the saliency of selected objects in the scene and distribute the computational resources according to the predicted visual attraction of those objects. As a consequence, the final result obtained by our approach will always be of high quality, even if the initial prediction has been somewhat unfortunate. In such a case, the only deficiency of our approach lies in more perceivable image artifacts during intermediate stages of image computation.

Eye ball tracking systems are occasionally included into specialized flight and combat training simulators [10]. Although they may be used to identify image regions that attract the user's attention, in this paper we discuss only those computational models of visual attention that do not require any specialized hardware.

3. Visual Attention Modeling

The amount of visual information reaching the eye is far beyond the processing capabilities of the human brain. In fact, only a small fraction of this information can be fully processed and assimilated into conscious experience [16]. After some initial information filtering performed in the retina and ganglion cells, the amount of information reaching the optic nerve falls into the range 10^8–10^9 bits per second [24]. Visual attention is an important means to adjust this amount of information to a manageable size.

Visual attention is often compared to a spotlight enhanc-

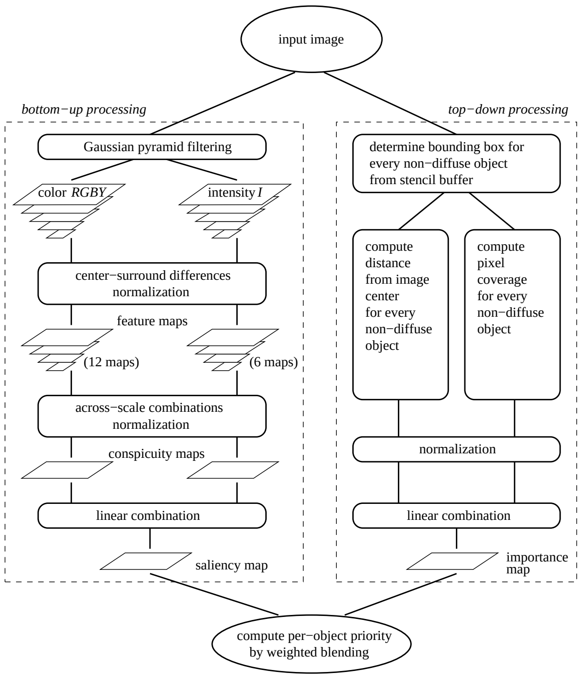

input image
bottom−up processing
top−down processing
Gaussian pyramid filtering
color RGBY
intensity I
center−surround differences normalization
feature maps
(12 maps)
(6 maps)
across−scale combinations normalization
conspicuity maps
linear combination
saliency map
determine bounding box for every non−diffuse object from stencil buffer
compute distance from image center for every non−diffuse object
compute pixel coverage for every non−diffuse object
normalization
linear combination
importance map
compute per−object priority by weighted blending

Figure 1: *General architecture of the attention model. The bottom-up component is a simplified version of the model developed by Itti et al.* [17]*. The top-down component was added to compensate for task-dependent user interaction.*

ing information in a selected region of the viewed scene. Shifting visual attention to a peripheral object can be performed with eye movements towards this object. This is referred to as overt attention, in which case attention is foveally centered by means of the saccadic (fast) fixation or smooth pursuit eye motion. For tasks that require only limited acuity, attention can be shifted without eye movements, which is referred to as covert attention. Systems relying on eye ball tracking [10] are unable to detect covert shifts of attention.

Early studies of eye movement trajectories (so called *scanpaths*) over images performed by Yarbus [46] and Noton and Stark [22] showed that subjects fixate similar regions of interest, although the order of attending these regions may differ significantly between subjects. Treisman developed the Feature Integration Theory (FIT) [37], which compares attention to the “glue” that integrates the separated features of an object such as shape, color, and motion. Recently, a “binary” theory of attention [4] gained significant popularity, which suggests that there are two main components of attention: *bottom-up* (rapid, scene dependent, object saliency-driven) and *top-down* (slow, task-driven, under cognitive control). The bottom-up derived saliency map selects locations that are most likely to be chosen for fixations. Unless these locations are completely irrelevant for the task at hand, the top-down processing usually selects among them. Many recent computational models of visual attention are strongly influenced by the FIT and binary theories [24, 16].

In our research, we choose a biologically plausible, state-of-the-art model of attention developed by Itti *et al.* [17]. In this paper we will only describe relevant aspects of this model. A detailed description is given in the original paper [17]. The model performs extremely well for static images [16] and was extensively validated by its developers in many demanding applications. An outline of its architecture is shown on the left side of Figure 1. An input image is decomposed into a set of separate channels such as intensity I or colors red, green, blue, and yellow (R, G, B, Y). Since we couldn’t find any significant differences in saliency prediction when including the orientations modality from the original Itti *et al.* model, we ignore this additional channel for the sake of computation efficiency. Each channel is decomposed into eight spatial frequency scales using Gaussian pyramids [5]. The chromatic opponency [30] is modeled for color channels by pairs $R-G$, $G-R$, $B-Y$, and $Y-B$ for each scale in the pyramids. Center-surround structures present in receptive fields of the retina, the lateral geniculate nucleus, and the primary visual cortex [30] are modeled as differences between low and high frequency scales. In fact, a so-called *color double-opponent* model is assumed in which pairs $R-G$ and $B-Y$ at the center of receptive fields are contrasted against pairs $G-R$ and $Y-B$ at the surround region of receptive fields, respectively. As the result of center-surround structure modeling, six feature maps for intensity and twelve feature maps for RG and BY opponent color channels are created. For each feature map, a normalization operator replicating cortical lateral inhibition mechanisms [30] is applied, whose main task is to promote maps with a small number of strong features and to suppress maps with numerous comparable features. The feature maps are then combined into two *conspicuity maps* for intensity and color by simple summation across all scales. According to the FIT theory [37], the conspicuity maps that represent different modalities are normalized and summed up into a so-called *saliency map*. Equal weights are assumed for each conspicuity map during this summation according to the hypothesis that similar features compete strongly for saliency, but different modalities contribute independently to the resulting saliency [17].

The attention model developed by Itti *et al.* was originally designed for static images. However, in interactive application scenarios, the user’s volitional focus of attention should be considered. A common observation is that the user tends to place objects of interest in the proximity of the image center and to zoom in on those objects in order to see them in more detail. In our approach, which is shown on the right side of Figure 1, we determine the bounding box for every non-diffuse object using the unique identification code in the stencil mask (see Section 4.1) and measure the distance between the bounding box center and the image center. We normalize the obtained distance with respect to half the length of the image diagonal. Additionally, we consider the object coverage in the image plane measured as the percentage of associated pixels in the stencil mask with respect to the number of pixels in the whole image. Although those two factors are often considered as the bottom-up saliency measures for static images [24], for interactive applications their meaning changes towards more task-driven top-down factors. We combine those two factors, and as a result we obtain the task-driven object *importance map*. Finally, we combine the importance map with the saliency map obtained from the bottom-up processing. A control slider is provided to the user to adjust the weights for balancing between saliency-driven and task-driven image correction. Figure 1 outlines the architecture of the complete attention model including our top-down extension to the original Itti model. Figure 4* shows an input image as it is submitted to our attention model and the resulting saliency map.

Saliency maps can be processed in two basically different ways to order corrective computations. The original Itti model includes a winner-takes-all strategy that chooses the most salient object for fixation. In this case, the most salient object will be corrected until convergence, before the next salient object is chosen for processing. However, as observed by Noton and Stark [22], there are many variations between subjects in the order of fixations along measured scanpaths. We found that in our application the second approach delivers better results: after the most salient object is roughly corrected, its saliency is reduced by a certain amount (e.g., 10 %) and the selection of the most salient object is repeated. This ordering approach allows other objects with a similar level of saliency to be corrected at least roughly before the most salient object is fully corrected.

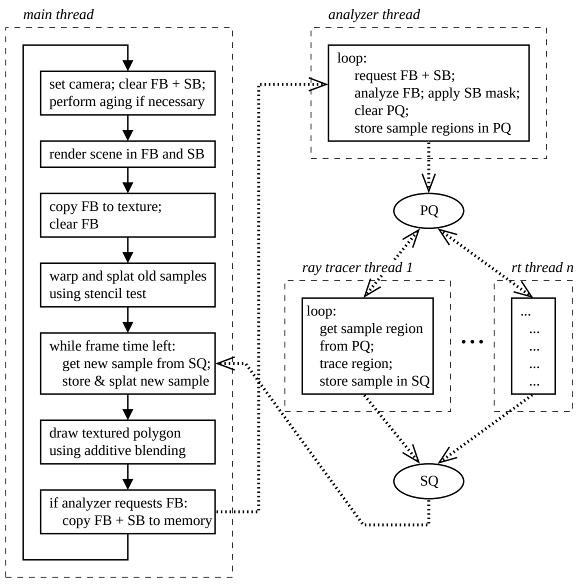
4. Corrective Splatting

In this section we describe the basic idea of our corrective splatting approach as well as some details of our method. We also comment on implementation issues where appropriate.

4.1. Overview

In a preprocessing step we compute a view-independent global illumination solution of our scene [39] and store the result (i.e., the illumination values) either in textures [21] or along with the vertices of triangle meshes.

After reading the scene description, we construct an object hierarchy that takes into account both the material properties and spatial distances between the primitives following the approach in [13]. Every group in this hierarchy contains primitives of the same material in a close spatial neighborhood. A scene that is, for instance, composed of a wooden table and two china cups on the table will be represented by a hierarchy where the root node consists of the table's primitives and two child nodes, each of them representing one of the cups, respectively. Thus the resulting hierarchy somehow



main thread
set camera; clear FB + SB; perform aging if necessary
render scene in FB and SB
copy FB to texture; clear FB
warp and splat old samples using stencil test
while frame time left: get new sample from SQ; store & splat new sample
draw textured polygon using additive blending
if analyzer requests FB: copy FB + SB to memory
analyzer thread
loop: request FB + SB; analyze FB; apply SB mask; clear PQ; store sample regions in PQ
PQ
ray tracer thread 1
loop: get sample region from PQ; trace region; store sample in SQ
rt thread n
...
SQ

Figure 2: *Data flow during rendering. The dotted lines depict data flow between different threads. The following abbreviations are used: FB = frame buffer, SB = stencil buffer, PQ = priority queue, SQ = sample queue. The number of ray tracing threads depends on the number of processors available (cf. Section 4.1).*

resembles the modeling hierarchy, which might not be available anymore after the preprocessing step. After the hierarchy has been constructed, we assign a unique identification code $id \in \{1, 2, 3, \dots\}$ to each group in the hierarchy that contains an object with non-diffuse material properties, e.g., specularity, reflectivity, or transparency. For every group that contains only purely diffuse objects, we set $id = 0$.

Next, we start one *analyzer thread*, which performs a perceptual analysis of the frame buffer (see Section 3), and at least one *ray tracing thread*. The rendering takes place within the *main thread* that was started by the user. Although our system also works on a mono-processor machine, it performs much more efficiently if at least three processors on a multi-processor machine are available. For every additional processor available, we start one additional ray tracing thread to increase the overall 'corrective performance'. The user can specify a desired frame rate f for the update of the display. During an interactive session, we perform the following steps for every frame that is displayed (see also Figure 2):

1. Get current wall clock time t_0.
2. If the camera has changed: set up new camera and perform aging of the cached samples (see Section 4.5).
3. Render the scene hierarchy using OpenGL graphics hardware. Lighting is disabled, since the color information stored in vertices and/or textures represents illumination

samples from a global illumination solution. During rendering, the unique identification code of every object is written into the stencil buffer for every pixel where the corresponding depth test passes.

4. Copy the content of the frame buffer into a texture T_{FB}. This operation is very fast, since it takes place entirely within the graphics hardware. Then clear the frame buffer, but keep the content of the stencil buffer.
5. Warp and splat samples from previous frames into the frame buffer with stencil test enabled, so that a sample that belongs to object *id* will only cover pixels with associated stencil value *id*.
6. Get current wall clock time t_1. While $t_1 - t_0 < 1/f$: collect samples from ray tracing thread(s), splat them into frame buffer and additionally store them for reuse, and update t_1.
7. Enable blending with blending function set to additive mode (`glBlendFunc(GL_ONE,GL_ONE)`), enable texturing using texture T_{FB}, switch to orthogonal projection, and draw a single rectangle of the size of the frame buffer. By setting minification and magnification filters to `GL_NEAREST`, a one to one correspondence of the pixels in the frame buffer and the texels in T_{FB} is obtained.
8. If the analyzer thread requests a new frame buffer image, copy frame buffer and stencil buffer to memory and inform the analyzer thread about new content.

As long as the rendering (step 3) and splatting (step 5) can be performed on the available graphics hardware within at most $1/f$ seconds, the frame rate f will roughly be kept. On current graphics hardware, the steps 4 and 7 are usually very fast: they sum up to about 0.05 milliseconds for a frame buffer of 512×512 pixels on our operating platform. Step 8 can be somewhat more costly (see Section 5 for detailed timings), but this step is typically performed every third or fourth frame only. The correction loop (step 6) is only performed as long as the frame time budget is not exceeded.

4.2. Inter-Thread Communication

As a consequence of our system's design, the analyzer thread and the ray tracing threads are running asynchronously to the main thread. Communication between the threads is accomplished by using mutual exclusion locks (mutexes) and condition variables to access common data structures such as the priority queue (PQ) or the sample queue (SQ), see Figure 2. In our implementation, we use the `pthreads` library as an API to the POSIX thread model.

The entries of the PQ are coordinates of sample regions in image space with an associated priority as computed by the analyzer thread. The analyzer thread may enter these sample regions in an arbitrary order w.r.t. their priorities. Any ray tracing thread that accesses the PQ automatically obtains the entry with the highest priority. Ray tracing threads may also resubmit entries to the PQ for load balancing, see Section 4.4. Every sample that is computed by one of the ray tracing threads is stored in the SQ. The SQ is a FIFO buffer, which is read during the correction loop (step 6 in the previous section) in the main thread.

4.3. Sampling and Splatting

Samples are generated using a (recursive) ray tracer, which employs importance sampling for area light sources and distributed ray tracing for glossy surfaces. Local illumination at hit points is evaluated using the shading models proposed by Cook-Torrance [7] and Ward [43].

As a result of the perceptual analysis of the frame buffer, a sample region in image space is submitted to the ray tracer. To efficiently create point samples that cover the sample region, we basically follow the approach in [33]. However, our sampling scheme differs from the previous one in that we iteratively compute point samples in image space instead of recursively sampling object space. This sampling scheme has the following advantages:

- In contrast to random sampling strategies, it is guaranteed that the number of samples needed for splatting into a sample region until convergence against a ray traced solution is exactly the same as for ray tracing the sample region.
- The sample positions are computed in such a way that the samples can be splatted efficiently using square footprints, such that the resulting image converges progressively against a ray traced solution.

The $\sqrt{5}$ sampling scheme from [33] uses a hierarchy of uniform grids, which are scaled by a factor of $w = 1/\sqrt{5}$ and rotated by an angle of $\alpha = \arctan 1/2 \approx 26.6°$ from one level to the next. The initial grid is spawned by the vectors $u_1 = (h, 0)$ and $v_1 = (0, h)$, i.e., it is aligned to the Cartesian coordinate system and has a grid step size of h. The spawning vectors of grid level ℓ are recursively defined as

$$u_{\ell+1} = \frac{2u_\ell + v_\ell}{5}, \quad v_{\ell+1} = \frac{-u_\ell + 2v_\ell}{5}.$$

For every grid level ℓ, the sample positions $p_{ij} \in \mathbb{R}^2$ are obtained as $p_{ij} = i \cdot u_\ell + j \cdot v_\ell$, $i, j \in \mathbb{Z}$.

To tailor this sampling scheme to our needs, we introduced the following changes:

1. The generation of sample positions is initialized by the desired level ℓ and the coordinates of the axis-aligned bounding box of the sample region.
2. The sample positions can be iterated efficiently in a non-recursive way.

In the initialization step we determine the min and max indices $j_{\min}$ and $j_{\max}$ of the sample positions p_{ij} of level ℓ that lie inside the sample region. Then we use the Cohen-Sutherland line clipping algorithm to compute the min and max indices $i_{\min}(j)$ and $i_{\max}(j)$ of the sample positions p_{ij} for $j_{\min} \leq j \leq j_{\max}$. While iterating through the sample positions, special care has to be taken to omit those positions that have already been generated for a coarser level. Due to the regular structure of the sampling pattern, the condition for omitting a sample position can easily be checked: if $(i+2j) \bmod 5$ is equal to zero, the sample position p_{ij} has to be omitted[†]. In this way, all sample positions $\{p_{ij} \mid j_{\min} \leq j \leq j_{\max}, i_{\min}(j) \leq i \leq i_{\max}(j)\}$ within the sample region can be enumerated efficiently: our iterative algorithm is about eight times faster than the original recursive algorithm.

Since the sample positions of level 1 are aligned to the coordinate system, the size s_1 of the square splat footprint is given by $s_1 = h$. For sample positions of level ℓ, the size s_ℓ can be computed as

$$s_\ell = \frac{h}{(\sqrt{5})^{\ell-1}} \max(|\cos((\ell-1)\,\alpha)|, |\sin((\ell-1)\,\alpha)|)\,.$$

Figures 5* and 6* illustrate the structure of the sampling pattern and the results from splatting the samples using a square splat footprint. This sampling pattern has the particularly nice property that the center point of a splat (i.e., the sample position) of any level ℓ will never be covered by any other splat of any level $\ell' > \ell$. If we make sure that splats are drawn level by level starting from level 1, the final image should converge to a pixel-precise ray traced image as ℓ is increased up to the level $\ell_{\max}$, for which $\mathrm{round}(s_{\ell_{\max}}) = 1$. Unfortunately, this is not true for a discrete image plane such as a raster screen: the sample positions of level $\ell_{\max}$ do in general not coincide with the pixel coordinates, so that due to integer rounding some pixels will be overwritten, while other pixels will remain blank. To overcome this problem, we rotate all grids by $-(\ell_{\max}-1)\,\alpha$. In this way, the finest grid is aligned to the coordinate system of the screen, i.e., the rows and columns of pixels. Now we only have to adjust the grid step size h such that the splat size $s_{\ell_{\max}} = h/(\sqrt{5})^{\ell_{\max}-1}$ is equal to one: $h = (\sqrt{5})^{\ell_{\max}-1}$. Using these modifications, the union of the splats of all sample positions from level one to level $\ell_{\max}$ covers all pixels of the image in such a way, that every pixels obtains the same color as in a ray traced solution.

4.4. Load Balancing

To ensure optimal utilization of multi-processor operating platforms, we employ a load balancing scheme among the ray tracing threads. To this end, the user specifies a job size

[†] This follows from the fact, that any sample position $p_{i'j'} = i' \cdot u_{\ell+1} + j' \cdot v_{\ell+1}$ from level $\ell+1$ has to be omitted, if it coincides with a sample position $i \cdot u_\ell + j \cdot v_\ell$ from level ℓ. Substituting the recursive definitions for $u_{\ell+1}$ and $v_{\ell+1}$, this condition is equivalent to $i' = (2i-j)/5$, $j' = (i+2j)/5$, since $u_\ell \perp v_\ell$. Since $i', j' \in \mathbb{Z}$, this reduces to $(i+2j) \bmod 5 = 0$.

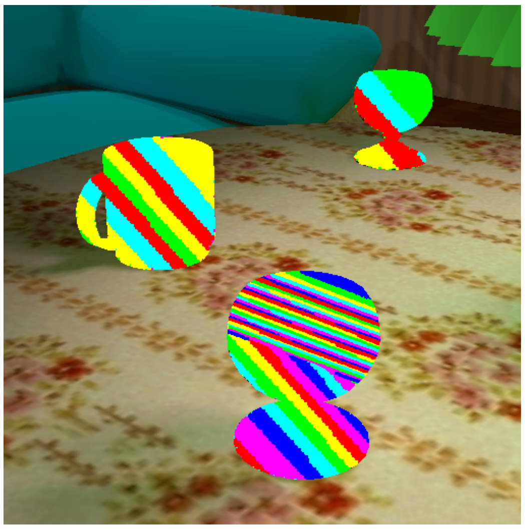


Figure 3: *Load balancing among six ray tracing threads. Three non-diffuse objects (two glasses and a mug) are placed on a table. The pixels of these objects are color-coded to indicate which of the six ray tracing threads computed the respective pixel.*

N_{job}, which represents the maximum number of samples that are computed by a ray tracer within one job.

In addition to the coordinates and the priority, the priority queue (PQ) entries contain a level counter ℓ and an iterator position i. For every sample region the analyzer thread submits to the PQ, these values are initialized to $\ell = 1$ and $i = 0$. Whenever a ray tracing thread takes an entry from the PQ, the number N_ℓ of sampling points in the associated level ℓ is computed. If $i + N_{\text{job}} < N_\ell$, a new entry with the same coordinates, priority, and level, but with the new iterator position $i + N_{\text{job}}$ is submitted to the PQ. If, on the other hand, $i + N_{\text{job}} \geq N_\ell$, the ray tracing thread can complete the level ℓ. If in this case $\ell < \ell_{\max}$, a new entry with the same coordinates, but with reduced priority, level $\ell+1$, and iterator position $i = 0$ is submitted to the PQ. We found that reducing the priority by 10–20 % seems appropriate to keep the balance of the initially computed priorities. Figure 3 illustrates the effect of load balancing by color-coding the sample positions with respect to the ray tracing thread.

4.5. Aging of Samples

The generation of samples is an expensive operation, especially if the scene contains many transparent and/or highly reflective objects and an expensive shading model is used. Thus we store samples in a cache for reuse and warp them before splatting according to the new camera settings, see also Figure 6*. To avoid high warping and splatting costs, we control the number of samples in our cache with an aging mechanism. In contrast to previous approaches [42, 32], our aging mechanism is based on a validity measure rather than using a constant rate aging. Since our samples contain only view-dependent illumination information, the validity of a sample can be measured by the deviation of the dot product between the surface normal and the current viewing direction with respect to the value of the dot product during sample generation.

Our camera model holds an age counter, which is initialized to zero and incremented whenever the camera settings change. When a sample is created by a ray tracer, the current age of the camera is stored along with the position and color of the sample. In addition, we store the (interpolated) surface normal of the hit point and the value of the dot product between this normal and the viewing direction for which the sample has been computed. Both the surface normal and the dot product are computed during the shading process, so that we do not need to perform additional computations.

At the beginning of every frame we perform aging of the samples in our cache if the settings of the camera c have changed (step 2 of the algorithm in Section 4.1). For every non-diffuse object o we determine the first level ℓ_0, for which there are at least $N_{\min}$ samples available. For every sample s of level ℓ_0 we compute the dot product d_{new} between the stored surface normal and the current viewing direction for that sample. If the absolute difference between d_{new} and the stored dot product value d of s is larger than a user-specified value $\delta \in [0,1]$, we mark the sample s as invalid. If more than half of the level-ℓ_0 samples of a certain age j are invalid, we delete all samples of age j of the object o. The following pseudo-code illustrates this aging mechanism once more:

```
for every non-diffuse object $o$
  determine starting level $\ell_0$
  for age $i = 0, \ldots, \texttt{MAX\_AGE}$
    init counter: $N_{\texttt{invalid}}[i] = N_{\texttt{total}}[i] = 0$
  for every level-$\ell_0$ sample $s$ of $o$
    set viewing direction $v = c.\texttt{eye} - s.\texttt{pos}$
    compute dot product $d_{\texttt{new}} = \langle s.\texttt{normal}, v \rangle$
    increase $N_{\texttt{total}}[c.\texttt{age} - s.\texttt{age}]$ by one
    if $(|d_{\texttt{new}} - s.d| > \delta)$ then
      increase $N_{\texttt{invalid}}[c.\texttt{age} - s.\texttt{age}]$ by one
  for level $\ell = 1, \ldots, \ell_{\texttt{max}}$
    for every level-$\ell$ sample $s$ of $o$
      set age difference $j = c.\texttt{age} - s.\texttt{age}$
      if $((j > \texttt{MAX\_AGE})$ or
          $(N_{\texttt{invalid}}[j] > 0.5 * N_{\texttt{total}}[j]))$ then
        delete $s$
```

In our simulations, we typically use $N_{\min} = 10$ and $\delta \in [0.05, 0.2]$. We store samples in a nested data structure of linked lists: every non-diffuse object has its own lists of samples for each level of hierarchy. This data structure permits efficient aging, which takes about 0.01 seconds for 50,000 cached samples in our implementation.

5. Results

The implementation of our approach has been integrated into our rendering system, which supplies sophisticated functionalities for OpenGL rendering, ray tracing, and shading. All timings in this section have been measured for an image size of 512×512 pixels on an SGI Onyx3 with eight 400 MHz R12k processors and 8 GB main memory. Our application runs also on a single processor SGI O2 and on a standard Linux PC. Due to the lack of multiple processors on these machines, however, the frame rates are not comparable to those obtained on the Onyx.

We performed a number of informal experiments with the visual attention model described in Section 3. We found that the predictions of the bottom-up component are usually in good agreement with the user's fixations at initial stages of interaction, when the user is not familiar with the scene. At later stages of interaction, we noticed that users more often deliberately tend to ignore the most salient objects and inspect less "attractive" scene regions. In this case, our simple top-down modeling compensated quite well for this goal-directed user behavior. We found that providing the user with the control over the weights assigned to the saliency- and task-driven attention components is very useful. For users familiar with the scene, the corrective splatting performed best when the two attention components were roughly balanced. As the result of our simplifications to the Itti model (see Section 3), the processing time for the complete attention model was below 0.3 seconds.

During an interactive session, the computation time for rendering each frame is made up by several components:

- aging of samples: 0.005–0.01 s for 50,000 samples;
- OpenGL rendering: 0.04–0.05 s for about 90,000 triangles with 100 different textures;
- warping and splatting cached samples: 0.02–0.03 s for 50,000 samples;
- reading frame and stencil buffer: 0.02 s.

Taking into account that reading the frame and stencil buffer happens about every 0.3 seconds, the total frame rate obtained from these timings is about 10–14 fps. This frame rate does not include the correction step of the rendering loop. If the user sets the desired frame rate to, e.g., 8 fps, there are on the average about 0.025–0.05 seconds left for correction within each frame. During 0.05 seconds, we can generate and splat the following numbers of corrective samples:

# ray tracing processors	# samples
2	500–800
4	1000–1500
6	1500–2500

The numbers depend on the complexity of the scene, the ray tracing depth, and the number and material properties of the objects that are hit during ray tracing.

Clearly, the OpenGL rendering and the warping and splatting of cached samples are the bottleneck of our implementation. These two steps should definitely benefit from adapting our data structures to the graphics hardware. Currently, we do not make use of triangle strips or fans for rendering. The generation of such connected groups of triangles could be integrated seamlessly into the preprocessing step. Also, the splatting of old samples could be sped up by using vertex arrays. However, the rendering bottleneck is somehow attenuated by the fact that due to our mesh-to-texture preprocessing [21], we were able to reduce the triangle count in our scenes by up to 80 % compared to the number of triangles that had been generated by adaptive subdivision during the precomputation of the global illumination solution.

6. Conclusion and Future Work

We introduced a novel approach for high quality rendering during interactive walkthroughs in environments containing objects with arbitrary light scattering characteristics. Due to the restricted computation time available during interactive rendering, our approach incorporates several aspects to obtain the best image quality as perceived by a human observer. To this end, we apply an advanced visual attention model to identify and order the image regions that are likely to be attended by the user. We introduced the concept of corrective splatting, which uses that order to generate ray traced point samples that are splatted into the image plane using a stencil buffer test. The sample positions are determined by a hierarchical sampling scheme, which has been adjusted to guarantee convergence to a fully ray traced solution. As a result of our processing, potential shading artifacts during camera motion are more likely to appear in less salient regions and will therefore be considered less annoying by the user. To enhance the visual appearance in uncorrected regions, we maintain a sample cache for reprojecting samples from previous frames. The validity of such old samples is based on a novel criterion, which is used for cache management.

Our implementation delivers good results when interactively navigating through scenes of medium complexity at about 10 fps. In contrast to other approaches that require (massively) parallel computations such as for instance [25, 42], our method performs very well on operating platforms with a limited number (4–8) of processors. Moreover, we can display high quality soft shadows and indirect illumination and we do not introduce any kind of rendering artifacts for purely diffuse objects. In addition to some optimizations regarding the OpenGL rendering, we would like to combine our approach with some level-of-detail and occlusion culling techniques to speed up rendering for scenes of higher geometric complexity. Moreover, it would be very interesting to investigate the applicability of global illumination methods that can handle dynamic environments. Finally, we would like to experiment with panoramic projection systems, where the user can only observe a limited area of the display at a time.

7. Acknowledgments

We would especially like to thank Marko Jung for setting up a new Onyx3 within record time. We also thank Christian Rössl for providing his user-friendly thread library and Marc Stamminger for pointing our attention to the $\sqrt{5}$ sampling pattern. Thanks also to all our colleagues who gave helpful comments and suggestions. This work was supported in part by the ESPRIT Open LTR Project #35772 SIMULGEN "Simulation of Light for General Environments".

References

1. R. Bastos, K. Hoff, W. Wynn, and A. Lastra. Increased photorealism for Interactive Architectural Walkthroughs. In *1999 Symposium on Interactive 3D Graphics*, pages 182–190, 1999.
2. M. R. Bolin and G. W. Meyer. A Frequency Based Ray Tracer. In *Computer Graphics (SIGGRAPH '95 Conf. Proc.)*, pages 409–418, 1995.
3. M. R. Bolin and G. W. Meyer. A Perceptually Based Adaptive Sampling Algorithm. In *Computer Graphics (SIGGRAPH '98 Conf. Proc.)*, pages 299–310, 1998.
4. J. Braun. Visual Search Among Items of Different Salience: Removal of Visual Attention Mimics a Lesion in Extrastriate Area V4. *Journal of Neuroscience*, 14(2):554–567, 1994.
5. P. J. Burt and E. H. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4):532–540, April 1983.
6. B. Cabral, M. Olano, and Ph. Nemec. Reflection Space Image Based Rendering. In *Computer Graphics (SIGGRAPH '99 Conf. Proc.)*, pages 165–170, 1999.
7. R. L. Cook and K. E. Torrance. A Reflectance Model for Computer Graphics. *ACM Trans. on Graphics*, 1(1):7–24, January 1982.
8. X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey. Multi-Layered Impostors for Accelerated Rendering. In *Computer Graphics Forum (Proceedings Eurographics '99)*, volume 18, pages C61–C72, 1999.
9. P. J. Diefenbach and N. I. Badler. Multi-pass Pipeline Rendering: Realism For Dynamic Environments. In *1997 Symposium on Interactive 3D Graphics*, pages 59–70, 1997.
10. A. Duchowski. Eye-based Interaction in Graphical Systems: Theory & Practice. *SIGGRAPH 2000 Course Notes #5*, 2000.
11. J. A. Ferwerda, S. Pattanaik, P. Shirley, and D. P. Greenberg. A Model of Visual Masking for Computer Graphics. In *Computer Graphics (SIGGRAPH '97 Conf. Proc.)*, pages 143–152, 1997.
12. A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers, San Francisco, CA, 1995.
13. J. Haber, M. Stamminger, and H.-P. Seidel. Enhanced Automatic Creation of Multi-Purpose Object Hierarchies. In *Proc. Pacific Graphics 2000*, pages 52–61, 2000.
14. W. Heidrich and H.-P. Seidel. Realistic, Hardware-Accelerated Shading and Lighting. In *Computer Graphics (SIGGRAPH '99 Conf. Proc.)*, pages 171–178, 1999.

15. H. Hoppe. View-Dependent Refinement of Progressive Meshes. In *Computer Graphics (SIGGRAPH '97 Conf. Proc.)*, pages 189–198, 1997.

16. L. Itti. *Models of Bottom-Up und Top-Down Visual Attention*. Ph.D. Thesis, California Institute of Technology, 2000. See also interactive demonstration at http://ilab.usc.edu/bu/.

17. L. Itti, C. Koch, and E. Niebur. A model of Saliency-Based Visual Attention for Rapid Scene Analysis. *Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

18. D. Lischinski and A. Rappoport. Image-Based Rendering for Non-Diffuse Synthetic Scenes. In *Rendering Techniques '98 (Proc. 9th EG Rendering Workshop)*, pages 301–314, 1998.

19. P. W. C. Maciel and P. Shirley. Visual Navigation of Large Environments Using Textured Clusters. In *1995 Symposium on Interactive 3D Graphics*, pages 95–102, 1995.

20. D. P. Mitchell. Generating Antialiased Images at Low Sampling Densities. In *Computer Graphics (SIGGRAPH '87 Conf. Proc.)*, volume 21, pages 65–72, 1987.

21. K. Myszkowski and T. L. Kunii. Texture Mapping as an Alternative for Meshing During Walkthrough Animation. In G. Sakas, P. Shirley, and S. Müller, editors, *Photorealistic Rendering Techniques*, pages 389–400. Springer–Verlag, 1995.

22. D. Noton and L. Stark. Scanpaths in the Eye Movements During Pattern Perception. *Science*, 171(968):308–311, 1971.

23. E. Ofek and A. Rappoport. Interactive Reflections on Curved Objects. In *Computer Graphics (SIGGRAPH '98 Conf. Proc.)*, pages 333–342, 1998.

24. W. Osberger. *Perceptual Vision Models for Picture Quality Assessment and Compression Applications*. Ph.D. Thesis, Queensland University of Technology, 1999.

25. S. Parker, W. Martin, P.-P. Sloan, P. Shirley, B. Smits, and C. D. Hansen. Interactive Ray Tracing. In *1999 Symposium on Interactive 3D Graphics*, pages 119–126, 1999.

26. F. Pighin, D. Lischinski, and D. H. Salesin. Progressive Previewing of Ray-Traced Images Using Image Plane Discontinuity Meshing. In *Rendering Techniques '97 (Proc. 8th EG Rendering Workshop)*, pages 115–126, 1997.

27. M. Ramasubramanian, S. N. Pattanaik, and D. P. Greenberg. A Perceptually Based Physical Error Metric for Realistic Image Synthesis. In *Computer Graphics (SIGGRAPH '99 Conf. Proc.)*, pages 73–82, 1999.

28. G. Schaufler. Dynamically Generated Impostors. In *Proc. of GI Workshop MVD'95*, pages 129–136, 1995.

29. M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *Computer Graphics (SIGGRAPH '92 Conf. Proc.)*, volume 26, pages 249–252, 1992.

30. R. Sekuler and R. Blake. *Perception*. McGraw-Hill, New York, 1994.

31. J. W. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In *Computer Graphics (SIGGRAPH '96 Conf. Proc.)*, pages 75–82, 1996.

32. M. Simmons and C. H. Séquin. Tapestry: A Dynamic Mesh-based Display Representation for Interactive Rendering. In *Rendering Techniques 2000 (Proc. 11th EG Rendering Workshop)*, pages 329–340, 2000.

33. M. Stamminger and G. Drettakis. Interactive Sampling and Rendering for Complex and Procedural Geometry. submitted. available at `http://www-imagis.imag.fr`.

34. M. Stamminger, J. Haber, H. Schirmacher, and H.-P. Seidel. Walkthroughs with Corrective Texturing. In *Rendering Techniques 2000 (Proc. 11th EG Rendering Workshop)*, pages 377–388, 2000.

35. M. Stamminger, A. Scheel, X. Granier, F. Perez-Cazorla, G. Drettakis, and F. Sillion. Efficient Glossy Global Illumination with Interactive Viewing. *Computer Graphics Forum*, 19(1):13–25, March 2000.

36. W. Stürzlinger and R. Bastos. Interactive Rendering of Globally Illuminated Glossy Scenes. In *Rendering Techniques '97 (Proc. 8th EG Rendering Workshop)*, pages 93–102, 1997.

37. A. Treisman and G. Gelade. A Feature Integration Theory at Attention. *Cognitive Psychology*, 12(1):97–136, 1980.

38. T. Udeshi and C. D. Hansen. Towards Interactive, Photorealistic Rendering of Indoor Scenes: A Hybrid Approach. In *Rendering Techniques '99 (Proc. 10th EG Rendering Workshop)*, pages 63–76, 1999.

39. V. Volevich, K. Myszkowski, A. Khodulev, and E. A. Kopylov. Using the Visible Differences Predictor to Improve Performance of Progressive Global Illumination Computations. *ACM Trans. on Graphics*, 19(2):122–161, April 2000.

40. J. R. Wallace, M. F. Cohen, and D. P. Greenberg. A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods. In *Computer Graphics (SIGGRAPH '87 Conf. Proc.)*, volume 21, pages 311–320, 1987.

41. B. Walter, G. Alppay, E. P. F. Lafortune, S. Fernandez, and D. P. Greenberg. Fitting Virtual Lights for Non-Diffuse Walkthroughs. In *Computer Graphics (SIGGRAPH '97 Conf. Proc.)*, pages 45–48, 1997.

42. B. Walter, G. Drettakis, and S. Parker. Interactive Rendering using the Render Cache. In *Rendering Techniques '99 (Proc. 10th EG Rendering Workshop)*, pages 19–30, 1999.

43. G. J. Ward. Measuring and Modeling Anisotropic Reflection. In *Computer Graphics (SIGGRAPH '92 Conf. Proc.)*, volume 26, pages 265–272, 1992.

44. G. J. Ward. The RADIANCE Lighting Simulation and Rendering System. In *Computer Graphics (SIGGRAPH '94 Conf. Proc.)*, pages 459–472, 1994.

45. G. Ward Larson. The Holodeck: A Parallel Ray-Caching Rendering System. In *Proc. of the 2nd Eurographics Workshop on Parallel Graphics and Visualization*, pages 17–30, 1998.

46. A. L. Yarbus. *Eye Movements and Vision*. Plenum Press, New York, 1967.

47. Y. L. H. Yee. *Spatiotemporal Sensitivity and Visual Attention for Efficient Rendering of Dynamic Environments*. M.Sc. thesis, Cornell University, 2000.

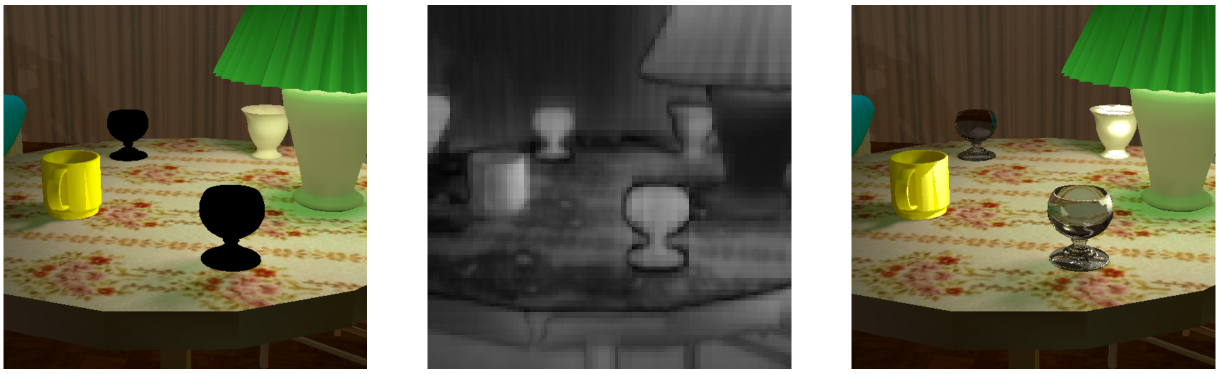


Figure 4: *Visual attention processing. Left: An input image showing a view-independent global illumination solution, which has been computed in a preprocessing step. Middle: The resulting saliency map, where bright grey levels encode objects with strong saliency. Right: The fully converged solution after corrective splatting has been applied according to the results of our attention model.*

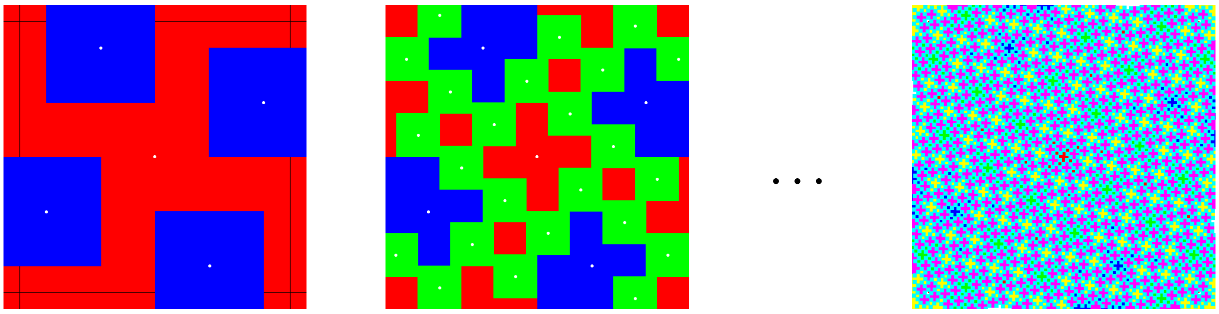


Figure 5: $\sqrt{5}$ *sampling scheme. All images are zoomed to the same level. Left: Level 1 splats (red) with edge length h cover the image domain. The size of the level 2 splats (blue) is* $\frac{h}{\sqrt{5}}\cos(\alpha)$, *cf. Section 4.3. Middle: Additional level 3 splats are shown in green. The center pixels of all splats (i.e., the sample positions) are marked in white. During convergence, these pixels will not be covered by any other splats. Right: Splats of levels 1–6 have been applied.*

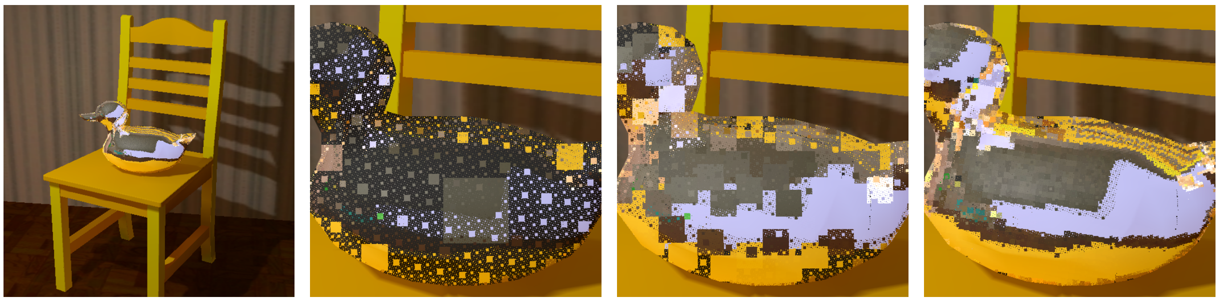


Figure 6: *Warping old samples for reuse. Left to right: a) Completely corrected view of a chrome-plated duck on a chair. b) Warping the samples from image a) for zooming in without computing new samples reveals the structure of the sampling pattern. c) Same as in b) plus new samples of level one and two (58 samples) splatted into the frame buffer. d) Same as in c) plus new samples of level three (188 samples). During an interactive session, the user will typically see corrected images of at least level three, since this correction level can be achieved with a few hundred samples per frame.*