A Survey on Temporal Coherence Methods in Real-Time Rendering

Daniel Scherzer[1,2] Lei Yang[3] Oliver Mattausch[2] Diego Nehab[4] Pedro V. Sander[3] Michael Wimmer[2] Elmar Eisemann[5]

[1]LBI for Virtual Archeology [2]Vienna University of Technology [3]Hong Kong UST [4]IMPA [5]Télécom ParisTech/CNRS-LTCI

Abstract

Nowadays, there is a strong trend towards rendering to higher-resolution displays and at high frame rates. This development aims at delivering more detail and better accuracy, but it also comes at a significant cost. Although graphics cards continue to evolve with an ever-increasing amount of computational power, the processing gain is counteracted to a high degree by increasingly complex and sophisticated pixel computations. For real-time applications, the direct consequence is that image resolution and temporal resolution are often the first candidates to bow to the performance constraints (e.g., although full HD is possible, PS3 and XBox often render at lower resolutions).

In order to achieve high-quality rendering at a lower cost, one can exploit temporal coherence *(TC). The underlying observation is that a higher resolution and frame rate do not necessarily imply a much higher workload, but a larger amount of redundancy and a higher potential for amortizing rendering over several frames. In this state-of-the-art report, we investigate methods that make use of this principle and provide practical and theoretical advice on how to exploit temporal coherence for performance optimization. These methods not only allow incorporating more computationally intensive shading effects into many existing applications, but also offer exciting opportunities for extending high-end graphics applications to lower-spec consumer-level hardware. To this end, we first introduce the notion and main concepts of TC, including an overview of historical methods. We then describe a key data structure, the so-called* reprojection cache*, with several supporting algorithms that facilitate reusing shading information from previous frames, and finally illustrated its usefulness in various applications.*

1 Introduction

In order to satisfy the ever increasing market demand for richer gaming experiences, developers of real-time rendering applications are constantly looking for creative ways to fit increased photo-realism, framerates, and resolution within the computational budget offered by each new graphics-hardware generation. Although graphics-hardware evolved remarkably in the past decade, the general sense is that, at least in the foreseeable future, any hardware improvement will be readily be put to use toward one of these goals.

The immense computational power required to render a single frame with desirable effects such as physically correct shadows, depth-of-field, motion-blur, and global illumination (or even an effective ambient-occlusion approximation) is multiplied by the demands of high-resolutions displays, which require large scene descriptions to be manipulated (geometry, textures). The difficulty is compounded further by the need to generate such frames continuously, as part of real-time animation.

Although rendering at 30Hz (NTSC) is already considered real-time, most modern LCD monitors and TVs can refresh at least at 60Hz. Naturally, developers strive to meet this standard. Given that there is still a measurable task-performance improvement in interactive applications as framerates increase up to 120Hz [DER*10b], there is justification to target such high framerates. In this case, as little as 8 milliseconds are available to produce each complete photo-realistic image, and all involved calculations (including physical simulations and other tasks unrelated to rendering itself) have to fit within this time budget. Needless to say, this poses a difficult task.

The traditional approach to optimization in the context of real-time rendering is to focus on improving the performance of individual rendering tasks, one at a time. In this State of The Art Report, we present results that are connected by a

© The Eurographics Association 2011.

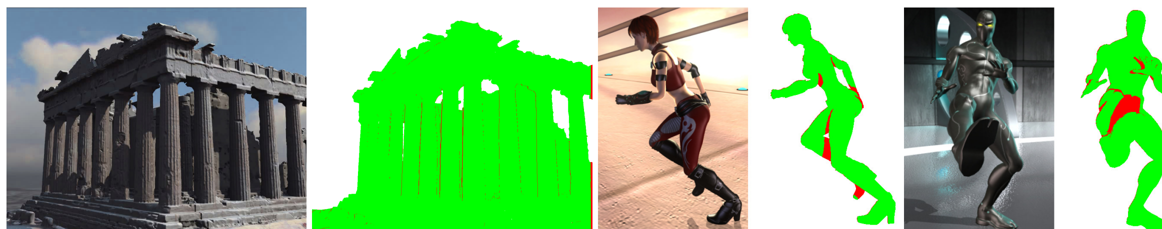

Figure 1: *Real-time rendering applications exhibit a considerable amount of spatio-temporal coherence. This is true for camera motion, as in the Parthenon sequence (left), as well as animated scenes such as the Heroine (middle) and Ninja (right) sequences. Diagrams to the right of each rendering show disoccluded points in red, in contrast to points that were visible in the previous frame, which are shown in green (i.e., green points are available for reuse). [Images courtesy of Advanced Micro Devices, Inc.]*

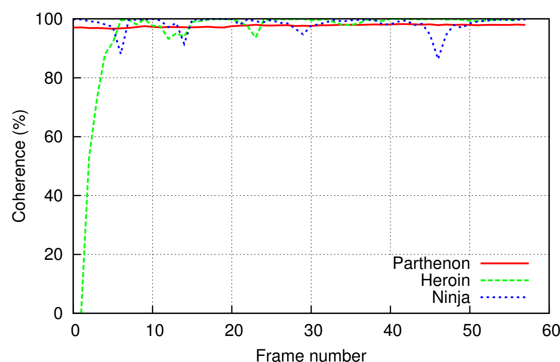

Figure 2: *Plot shows the percentage of surface points that remain visible from one frame to the next for the animations of Figure 1. Coherence in excess of 90% is typical of many game-like scenes.*

more general approach to optimization: exploiting temporal coherence (TC).

Consider the examples shown in Figure 1. When framerates are high, very little changes from one frame to the next. Each visible surface point tends to remain visible across the interval of several frames. Furthermore, points attributes (including color) tend to maintain their values almost unchanged throughout. To measure the amount of TC in these animation sequences, Figure 2 plots the fraction of points that remain visible from one frame to the next. We can see that fractions of 90% and higher are typical.

Since an ever increasing slice of the rendering budget is dedicated to shading surface points, such high levels of TC presents a great opportunity for optimization. Rather than wastefully recomputing every frame in its entirety from scratch, we can reuse information computed during the course of one frame (intermediate results, or even final colors) to help render the following frames. The resulting reduction in the average cost of rendering a frame can be used in a variety of ways: from simply increasing the framerate to improving the quality of each rendered frame.

Naturally, TC has been exploited since the early days of computer graphics. We describe a variety of early applications in Section 2. In Section 3, we move to methods that can be used to take advantage of TC in real-time rendering scenarios. Special attention is given to techniques based on *reprojection*. Reprojection allows us to map a surface point in one frame to the same surface point in a previously rendered frame. Needless to say, this mapping plays a key role in the reuse of information across frames. Reusing information involves certain quality/performance trade-offs that are analyzed in Section 4. Since the selection of a proper target for reuse can modulate this trade-off, the same section discusses the most important factors influencing this choice. Using Sections 3 and 4 as a basis, we then describe many applications that take advantage of TC in Section 5. Finally, in Section 6, we provide a comparison, an outlook on future directions and a summary of the presentation.

2 Early approaches

This state-of-the-art report concentrates on recent real-time rendering methods that exploit temporal coherence by reusing information created previously. On the other hand, temporal coherence has been around for almost as long as computer graphics itself. For example, the term *frame-to-frame coherence* was first introduced by Sutherland et al. [SSS74] in his seminal paper "Characterization of Ten Hidden-Surface Algorithms." Therefore, we will summarize early developments in which TC was already used in similar ways.

In particular, we will cover the use of temporal coherence in *ray tracing*, in *image-based rendering*, and *image and render caches*.

2.1 Ray-tracing

Temporal coherence was already used for the classical ray tracing algorithm in order to speed up the calculation of animation sequences. While these techniques are for offline rendering, most of them already make use of forward reprojection (Section 3.2) for reusing information.

Badt [BJ88] develop a forward reprojection algorithm that uses object space information stored from the previous frame. This allows approximating ray-traced animation frames of

diffuse polygons. Adelson and Hodges [AH95] later extend the approach to ray-tracing of arbitrary scenes. Havran et al. [HBS03] reuse ray/object intersections in ray casted walk-throughs. They do this by reprojecting and splatting visible point samples from the last frame into the current, thereby avoiding the costly ray traversal for more than 78% of the pixels in their test scenes.

Leaving the concept of frame-based rendering behind, Bishop et al. [BFMZ94] introduce frameless rendering, which heavily relies upon TC for sensible output. Here each pixel is rendered independently based on the most recent input, thereby minimizing lag. There is no wait period till all pixels of a frame are drawn, but individual pixels stay visible for a random time-span, until they are replaced with an updated pixel. Note that this approach does not use the object coherency that is such an integral part of many polygon renderers. To avoid image tearing pixels are rendered in a random order.

2.2 Image-based rendering

In the widest sense, temporal coherence also covers methods that replace parts of a scene with image-based proxy representations. This can be interpreted as a form of reverse reprojection (Section 3.1) applied to individual parts of a scene. This idea was used most prominently in the so-called hierarchical image cache and its variations [Sch96, SLS*96], where images (called impostors) of complex distant geometry are generated on the fly and reused in subsequent frames, thus reducing rendering times. The geometric error for such systems has also been formally analyzed [ED07a]. Frame-to-frame coherence is further exploited in various systems that partition the scene into different layers [RP94, LS97], while others augment the image-based representation with depth information [SGHS98]. In this report however, we will focus on methods that do not use proxy geometry to cache information, but directly reuse rendered information from the previous frame buffers.

2.3 Image and render caches

Image and render caches store the information generated in previous frames in a data structure, and reuse this information for the generation of the current frame, using different reconstruction and mostly forward reprojection techniques (Section 3.2).

Wimmer et al. [WGS99] accelerate the rendering of complex environments by splitting the scene into a near field and a far field: The near field is rendered using the traditional rendering pipeline, while ray casting is used for the far field. To minimize the number of rays cast, they use a *panoramic image cache* and only recompute rays if a cache entry is not valid anymore, where validity is based on the distance to the original observer position where the pixel was generated. The panoramic image cache avoids reprojection altogether, but quickly becomes inaccurate for translational motion.

Qu et al. [QWQK00] use image warping to accelerate ray-casting. The idea is to warp the output image of the previous frame into the current frame using forward projection. Due to the warping, pixels may fall between the grid positions of the pixels of the current frame, therefore an offset buffer is used to store the exact positions. Due to disocclusions, holes can occur at some pixels. Here ray-casting is used to generate these missing pixels. The authors propose to use an age stored with each pixel, which is increased with each warping step to account for the lower quality of pixels that have been warped (repeatedly). Upon rendering a new output frame, this age can be used to decide if a pixel should be re-rendered or reused.

Walter et al. [WDP99] introduce the *render cache*. It is intended as an acceleration data structure for renderers that are too slow for interactive use. In contrast to the previously mentioned approaches, which store pixel colors, the render cache is a point-based structure, which stores the complete 3d coordinates of rendered points and shading information. By using reverse reprojection, these results can be reused in the current frame. Progressive refinement allows decoupling the rendering and display frame rates, enabling high interactivity. Walter et al. [WDG02] extend this approach with predictive sampling and interpolation filters, while later work also accelerates the render cache on the GPU [VALBW06, ZWL05].

Simmons and Sequin [SS00] use the graphics hardware for reconstructing images from reprojected and new samples by interpreting the samples as vertices of a spherical mesh, called a tapestry.

3 Reprojection and data reuse

An important decision when utilizing TC is how the previously computed data is stored, tracked, retrieved and reused. On modern graphics hardware, the most common way is to store the desired data at visible surface points in viewport-sized off-screen buffers at each rendered frame, usually referred to as *history buffer*, *payload buffer* or *cache*. When generating the following frames, the data in the buffer are reprojected to their new locations based on scene motion. Even with hardware support, reprojection can still be a computationally challenging task. In this section, we first describe two reprojection strategies that are commonly used in numerous applications and can sometimes be interchanged to suit special needs. In disoccluded regions where the previous data is not available, we show how to fill in approximate results that are visually plausible. Finally, we summarize different data reuse strategies that are commonly used in various applications described in Section 5.

3.1 Reverse reprojection

A basic scenario of using TC is to generate a new frame using data from a previously shaded frame. For each pixel in the new frame, we can trace back to its position in the earlier cached frame to determine if it was previously visible. If available, this cached value can be reused in place of performing an expensive computation. Otherwise it must be recomputed from scratch. This technique is called the *Reverse*

© The Eurographics Association 2011.

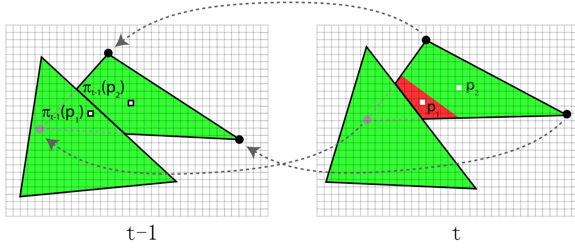

Figure 3: *The reverse reprojection operator. The shading result and pixel depths of time t-1 are stored in screen-space framebuffers (left). For each pixel $\mathbf{p}$ at time t (right), its reprojected position $\pi_{t\text{-}1}(\mathbf{p})$ is computed to locate the corresponding position at frame t-1. The recomputed scene depth is compared to the stored pixel depth. A pair of matching depths indicate a cache hit ($\mathbf{p}_2$), whereas inconsistent depths indicate a cache miss ($\mathbf{p}_1$).*

Reprojection Cache (RRC). It was proposed independently by Nehab et al. [NSL*07] and Scherzer et al. [SJW07], and serves as a framework for a number of applications described in Section 5.

Formally, let f_t denote the cache generated at time t, which is a framebuffer holding the pixel data visible at that frame. In addition to f_t, we keep an accompanying buffer d_t which holds the scene depth in screen space. Let $f_t(\mathbf{p})$ and $d_t(\mathbf{p})$ denote the buffer values at pixel $\mathbf{p} \in \mathbb{Z}^2$. For each pixel $\mathbf{p} = (x, y)$ at time t, we determine the 3D clip-space position of its generating scene point at frame t-1, denoted by $(x', y', z') = \pi_{t\text{-}1}(\mathbf{p})$. Here the reprojection operator $\pi_{t\text{-}1}(\mathbf{p})$ maps a point $\mathbf{p}$ to its previous position at frame t-1. Note that with this reprojection operation, we also obtain the depth of the generating scene point z' at frame t-1. This depth is used to test whether the current point was visible in the previous frame. If the reprojected depth z' equals $d_{t\text{-}1}(x', y')$ (within a given tolerance), we conclude that the current pixel $\mathbf{p}$ and the reprojected pixel $f_{t\text{-}1}(x', y')$ are indeed generated by the same surface point. In this case, the previous value can be reused. Otherwise no correspondence exists and we denote this by $\pi_{t\text{-}1}(\mathbf{p}) = \varnothing$, which we refer to as a *cache miss*. Additional tests such as object-ID equality can also be employed to reinforce this cache miss decision. The reverse reprojection operation is illustrated in Figure 3.

3.1.1 Implementation

The RRC algorithm can be conveniently mapped to the modern programmable rendering pipeline. A major task of this is to compute the reprojection operator $\pi_{t\text{-}1}(\mathbf{p})$, which maps each pixel $\mathbf{p}$ to its corresponding clip-space position in the previous frame $t-1$. At frame t, the homogeneous projection space coordinates $(x_t, y_t, z_t, w_t)_{vert}$ of each vertex $\mathbf{v}$ are calculated in the vertex shader, to which the application has provided the world, view and projection matrices and any animation parameters. To perform correct reprojection, the application also has to provide these matrices and animation parameters at $t-1$ to the vertex shading stage. In addition to transforming the vertex at frame t, the vertex shader also transforms the vertex using the matrices and parameters from frame $t-1$, thereby computing the projection-space coordinates $(x_{t\text{-}1}, y_{t\text{-}1}, z_{t\text{-}1}, w_{t\text{-}1})_{vert}$ of the same vertex at frame $t-1$. These coordinates are stored as vertex attributes and are automatically interpolated by the hardware before reaching the pixel stage. This gives each pixel $\mathbf{p}$ access to the previous projection-space coordinates $(x_{t\text{-}1}, y_{t\text{-}1}, z_{t\text{-}1}, w_{t\text{-}1})_{pix}$. The final cache coordinate $\pi_{t\text{-}1}(\mathbf{p})$ is obtained with a simple division by $(w_{t\text{-}1})_{pix}$ within the pixel shader. Note that the transformation only need to be computed at the vertex level, thereby significantly reducing the computational overhead in most scenes.

Because of arbitrary scene motion and animation, the previous position $\pi_{t\text{-}1}(\mathbf{p})$ usually lies somewhere between the set of discrete samples in the cache $f_{t\text{-}1}$ and thus some form of resampling is required. Nehab et al. [NSL*07] suggested using hardware-assisted bilinear texture fetch for resampling. In most situations this suffices for practical use. It is also used to reconstruct the previous depth value, so that a more robust cache-miss detection can be achieved.

3.2 Forward reprojection

Alternatively, instead of starting from every pixel in the target frame, we can directly process the cache and map every pixel in the cache into its new position. This has the advantage that it does not require processing the scene geometry for the new frame, which is desirable in some applications. Nevertheless, it requires a forward motion vector (or disparity vector) generated for each pixel, which is equivalent to the inverse mapping of $\pi_{t\text{-}1}(\mathbf{p})$.

Yu et al. [YWY10] propose a forward reprojection method that leverages the parallel data scattering functionality on the GPU (available through CUDA or DirectX 11 Compute Shader). For each pixel in the cache, they determine its new position in the target frame by offsetting its current position using the forward motion vector (disparity vector). Then the depth of the current pixel is tested against the target pixel for resolving visibility. This operation is performed using the atomic min functionality to avoid parallel write conflicts. Note that since there is no one-to-one mapping between the source and the target pixels, holes can be present after reprojection. To resolve this, Yu et al. [YWY10] propose to increase support size of the reprojected pixel, i.e. write to all four neighbors of each reprojected fractional position. This works with near-view warping for their light field generation, but may be insufficient for other applications where non-uniform motion is involved.

Per-pixel forward projection can be difficult and costly to implement on conventional graphics hardware (prior to DirectX 11). It may also require applying complex filtering strategies in order to acquire pixel-accurate results. A way around these problems was described by Didyk et

© The Eurographics Association 2011.

al. [DER*10b]: they proposed an image warping technique, which is efficient on conventional GPUs. The warping is achieved by approximating the motion vector field with a coarse grid representation, assuming that the vector field is piecewise linear. An initial uniform grid is snapped to large motion vector discontinuities in the previous frame. Then the grid geometry is rendered to its new position dictated by the motion vector field, so that its associated texture is automatically warped. Occlusion and disocclusion are naturally handled with grid folding and stretching. Note that depth testing must be enabled in order to correctly resolve occlusions and fold-overs.

A regular grid used by Didyk et al. [DER*10b] can have difficulties warping images with fine-detailed geometry. They later propose an improved algorithm using adaptive grids [DRE*10]. Their new approach starts with a regular grid (32×32). Then a geometry shader traverses all the quads in the grid in parallel. Every quad that contains a discontinuity is further partitioned in four. This process is iterated until no quads need to be further partitioned. At that point, the grid geometry is rendered as in the regular grid case. Due to the adaptive grid, this new approach has better utilization of computational resources, thereby significantly improving the quality.

3.3 Handling disocclusion

The process of reprojection is essentially a non-linear warping and may leave the newly disoccluded regions incorrectly shaded or blank. With reverse reprojection, we may have the option to reshade these regions whenever a cache miss occurs. However, this is not always desirable due to limited time budget or other constraints imposed by the application. With forward reprojection, there is usually no such option since the shading input may not be available. Therefore, some form of approximate hole filling needs to be performed in order to reduce visual artifacts.

Andreev [And10] suggests an inpainting strategy that duplicates and offsets neighboring image patches into the hole area from the four sides. This is efficiently implemented in a pixel shader and can be performed iteratively until the hole is completely filled. For a more robust solution, one can consider using pull-push interpolation [MKC07]. The pull-push algorithm consists of a pull phase and a subsequent push phase. The pull phase iteratively computes coarser levels of the image containing holes, forming an image pyramid. Each pixel in a coarser level is the average of the valid pixels in the corresponding four pixels from the finer level. The push phase then operates in the inverse order and interpolates the hole pixels from the coarser levels. This works best for the small holes caused by per-pixel forward reprojection. With larger holes, the interpolated pixels may appear blurred and can be a source of artifacts as well.

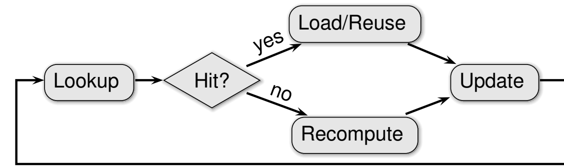

Figure 4: *Schematic diagram of applying the reverse reprojection cache to avoid pixel shading whenever possible [NSL*07].*

3.4 Cache refresh

A straightforward usage of data reprojection is to avoid shading pixels that are visible in the previous frame. This can apply to either part or the entire pixel shading computation. For example, if we use RRC, the original pixel shader can be modified to add a cache load and reuse branch, as shown in Figure 4. When each pixel $\mathbf{p}$ is generated, the reprojection shader fetches the value at $\pi_{t\text{-}1}(\mathbf{p})$ in the cache and tests if the result is valid (i.e. cache hit). If so, the shader can reuse this value in the calculation of the final pixel color. Otherwise, the shader executes the normal pixel shading. Whichever route the shader follows, it always stores the cacheable value for potential reuse during the following frame.

Although a cached value can be continuously reused throughout many frames, it may quickly become stale because of either shading changes or resampling error. Nehab et al. [NSL*07] propose to refresh (i.e. recompute) the value periodically in order to counteract this effect. For a fixed refresh rate, the screen can be divided into Δn groups and updated in a round-robin fashion in each frame by testing the following condition for each pixel:

$$(t+i) \quad \bmod \Delta n = 0, \tag{1}$$

where i is the group ID of the pixel and t is a global clock. They suggest two simple ways of dividing the screen:

Tiled refresh regions. The screen is partitioned into a grid of Δn non-overlapping tiles, with pixels in a tile sharing the same ID.

Randomly distributed refresh regions. The screen pixels are equally partitioned into Δn groups with each pixel assigned a random group ID.

Interleaved refresh regions. The updated screen pixels are uniformly distributed on a regular grid. For a static scene and camera, interleaving n such images leads to an accurate high resolution image of the scene.

With the tiled refresh strategy, pixels within a tile are refreshed at the same time. This leads to excellent refresh coherence, but may lead to visible discontinuity at tile boundaries. The randomly distributed refresh strategy updates pixels in a random pattern. It exchanges sharp discontinuities for high-frequency noise, which is usually less objectionable. Note that it is recommended to assign the same ID to each 2×2 or

© The Eurographics Association 2011.

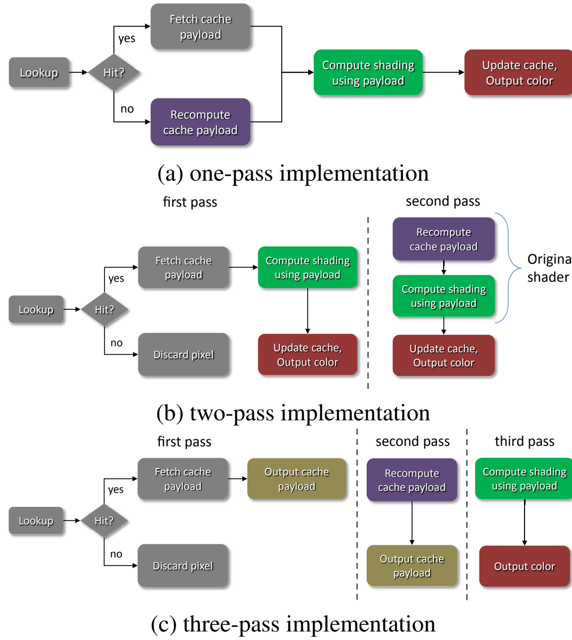

Figure 5: *Three control flow strategies for accelerating pixel shading using the RRC.*

larger quad of pixels, because modern GPU performs lock-step shading computation on such quads. The interleaved refresh regions are easy to achieve by rendering low-resolution frames and applying an distance-dependent offset on the geometry. For temporal integration such schemes are interesting, as the combination of these samples leads to a high-resolution shot.

In addition, care must be taken in order to maximize the performance when implementing this scheme with RRC. The fact that there are two distinct paths in Figure 4, i.e. cache hit and cache miss, allows for several implementation alternatives. The most straightforward approach is to branch between the two paths (Figure 5(a)). This allows all the tasks to be performed in a single rendering pass, but may suffer from dynamic branching inefficiency particularly when the refreshed region is not coherent and the branches are unbalanced. To achieve better performance, Nehab et al. [NSL*07] propose to defer the expensive recomputation and put it into a separate pass, so that the branches are more balanced (Figure 5(b)). By relying on early-Z culling, the miss shader is only executed on the cache-miss pixels that are automatically grouped to avoid any performance penalty. If the hit shader (green block in Figure 5) is also non-trivial to compute, the branches in the first pass may still not be balanced. Sitthi-amorn et al. [SaLY*08a] propose to further separate this part of the computation into a third pass (Figure 5(c)) in order to reduce dynamic branching cost. This three-pass implementation also has the advantage that it does not require muliple render-target support, but incurs more geometry processing cost. The choice of strategy therefore is dependent on the relative cost between vertex and pixel shading in the target scene. Sitthi-amorn et al. [SaLY*08a] give some empirical performance analysis of these three implementations in practice.

3.5 Amortized sampling

Another common strategy of data reuse is to combine previous shading results with the current one. Gradual phase-out can then be used to avoid explicit refreshing pixels. This strategy is usually applied to amortize the expensive task of computing a Monte-Carlo integral, in which multiple spatial samples are combined for each pixel. With data from the past, each frame then only needs to compute a lot less samples (typically only one) for each pixel in order to achieve a similar image quality. This is beneficial to many high-quality rendering effects described later, such as spatial anti-aliasing, soft shadow and global illumination.

In order to efficiently reuse and combine previously computed samples of a signal without increasing storage overhead, Nehab et al. [NSL*07] and Scherzer et al. [SJW07] propose to combine and store all previously computed samples associated with a surface point using a single running average. In each frame, only one sample $s_t(\mathbf{p})$ is computed for each pixel $\mathbf{p}$ and is combined with this running average using a recursive exponential smoothing filter:

$$f_t(\mathbf{p}) \leftarrow (\alpha)s_t(\mathbf{p}) + (1-\alpha)f_{t\text{-}1}\left(\pi_{t\text{-}1}(\mathbf{p})\right). \qquad (2)$$

Here the running estimate of frame t is represented by f_t and is stored in the RRC. If we expand this recursive formulation, we can see that the running estimate is equivalent to the weighted sum of all the previous samples at the same surface point. The weight of a single sample decreases exponentially over time, and the smoothing factor α regulates the tradeoff between the degree of variance reduction and responsiveness to changes in the sampled signal. For example, a small value of α leads to a relatively slow decrease of the sample weights, which effectively accumulates more samples in the past and therefore produces a smoother result at the expense of additional lag in the shaded signal.

The precise degree of variance reduction is given by

$$\lim_{t\to\infty} \frac{\mathrm{Var}\left(f_t(\mathbf{p})\right)}{\mathrm{Var}\left(s_t(\mathbf{p})\right)} = \frac{\alpha}{2-\alpha}. \qquad (3)$$

For example, choosing a value of $\alpha = 2/5$ reduces the variance to $1/4$ the original. This is roughly equivalent to combining 4 previous samples with equal weights. On the other hand, the actual number of frames contributing to f_t with non-trivial weights (i.e. larger than 8-bit precision $1/256$) is 10, which indicates that the contribution of any obsolete sample will be smoothed out after 10 frames. This tradeoff between smoothness and lag is illustrated in Figure 6. In practice, α must be carefully set to obtain the best tradeoff.

© The Eurographics Association 2011.

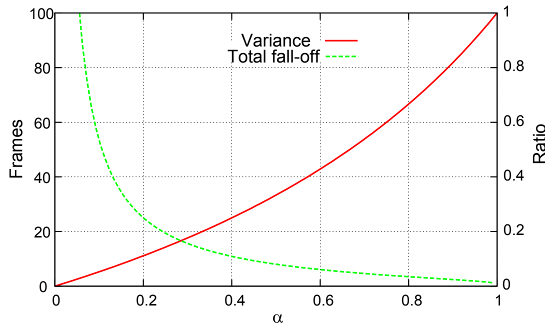

Figure 6: *Trade-off between the amount of variance reduction (the variance-ratio curve), and the maximum frames of lag that may exist in the current estimate (the total fall-off curve) [NSL*07]. This trade-off is controlled by the parameter α.*

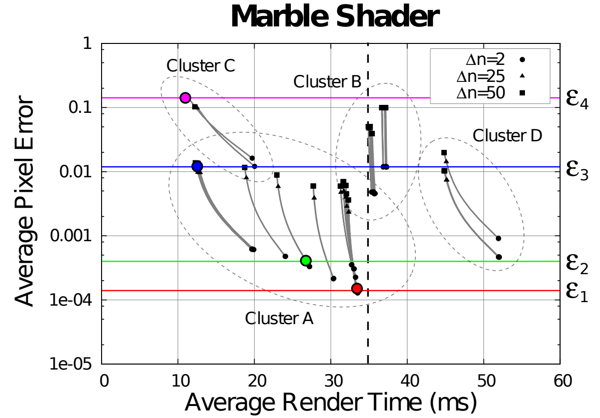

Figure 7: *Trade-off between error and performance associated to caching different intermediate results in a marble shader. Each line shows the effect of varying the refresh period Δn between 2 and 50 frames on each choice of cached intermediate computation. Interesting error thresholds ε_i are marked, the results of which are shown in Figure 8. Original shader runs at 29FPS, as indicated by the dashed line.*

4 Data reuse quality and performance

The ideal scenario for taking advantage of coherence is when the value of interest obtained from a previous frame is exactly the same as the desired one (i.e. were it recomputed from scratch). In reality, when considering a target for reuse, we often find that its value depends on inputs that are beyond our control. These may include changing viewing parameters, lighting conditions, time itself, and most importantly, user interactions. Good targets for reuse are those that change little under the range of expected input variation. Nevertheless, even slow varying attributes must be eventually updated, and we must also identify appropriate refresh periods.

Another important consideration is the cost of recomputing each reused value. This is because the overhead associated with obtaining previously computed values is not negligible (see Section 3.1.1). If recomputing a value is cheap, reusing it may not bring any performance advantage.

In summary, developers must identify computationally expensive intermediate computations that vary little under the range of expected input changes, and determine the appropriate number of frames between updates and reuse. Given the large number of different effects and input parameters involved in a modern real-time rendering application, this task can quickly become overwhelming. Recent efforts have therefore focused on automating parts of this process.

4.1 Semi-automatic target identification

The system proposed by Sitthi-Amorn et al. [SaLY*08b] starts by analyzing the source-code of shaders and identifying possible intermediate computations for reuse. During a training session, the system automatically renders animation sequences while gathering error and performance data on shaders that have been automatically modified to cache and reuse each candidate. The rendering sessions are designed to encompass the range of typical input variation, and are run under a variety of different refresh periods.

Assuming the input variation is stationary, the authors found empirical models for both the amount of error and the rendering cost associated to reusing each possible intermediate value. These models were later shown to closely match measurement data.

The expected error caused by reusing the value f_m of a given intermediate computation m over a period of Δn frames can be modelled by a parametric equation:

$$\hat{\varepsilon}(f_m, \Delta n) = \alpha_m \left(1 - e^{-\lambda_m(\Delta n - 1)}\right). \quad (4)$$

Parameters α_m and λ_m can be obtained by fitting the model to data gathered in the training session.

Modelling the cost of rendering each pixel requires more work. First, the system solves for an estimate of the average time taken to render a pixel under both cache-hit and cache-miss conditions. Denote these by $\Delta_{\text{hit}}(f_m)$ and $\Delta_{\text{miss}}(f_m)$, respectively. These values are obtained by solving an over-constrained linear system for $\Delta_{\text{hit}}(f_m)$ and $\Delta_{\text{miss}}(f_m)$:

$$H_i \Delta_{\text{hit}}(f_m) + M_i \Delta_{\text{miss}}(f_m) + c = \Delta t_i. \quad (5)$$

Each equation comes from measurements of different frames i in the training sequence. Here, c is a constant rendering overhead, H_i is the number of hits, M_i the number of misses, and Δt_i the time to render frame i.

The average cost of rendering a single pixel can then be modelled as

$$\hat{r}(f_m, \Delta n) = \lambda(\Delta n)\,\Delta_{\text{hit}}(f_m) + \big(1 - \lambda(\Delta n)\big)\,\Delta_{\text{miss}}(f_m), \quad (6)$$

© The Eurographics Association 2011.

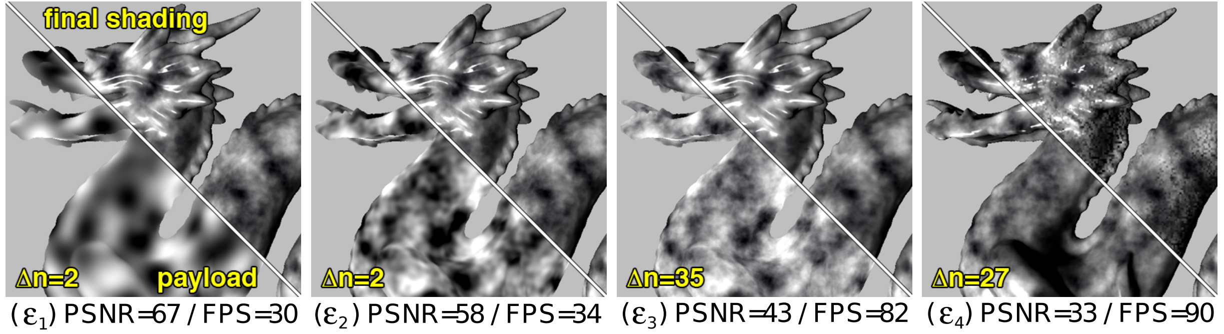

Figure 8: *Results of selecting different error thresholds in Figure 7. The intermediate value selected for caching (payload) is shown next to the final rendered results (final shading). Higher error thresholds allow for substantial parts of the computation to be cached, leading to better performance at the expense of quality.*

where $\lambda(\Delta n) = \mu(1 - 1/\Delta n)$ is an empirical model for the cache hit-rate as a function of Δn, and μ is obtained by fitting this model to the training data.

Using these models, the system allows the developer to specify a target average pixel error. It then automatically selects the shader component that provides the greatest improvement in performance without exceeding the error threshold.

Figure 7 shows the error/performance behavior associated to caching several different intermediate computations performed by a marble shader. This shader combines a marble-like albedo modeled as five octaves of a 3D Perlin noise function, with a simple Blinn-Phong specular layer. Figure 8 show the results of rendering under each choice of error tolerance, in terms of both quality and performance. As the user selects larger error thresholds, the system reacts by selecting larger portions of the computation for caching (see the payload), eventually including even the view-dependent lighting, at which point undesirable artifacts appear. Nevertheless, substantial performance improvements are possible below an acceptable error threshold (see e_3 running at a 2.8x improvement).

4.2 Reprojection errors and their accumulation

The strategies we use to obtain the values of previous computations (see Section 3) can themselves inject unwanted errors. Although such errors are indirectly modelled by the automatic method described above, here we present a simplified analysis of this specific issue (see [YNS*09] for an alternative, more detailed presentation).

Due to camera and object motions, the corresponding positions of any given surface point in two consecutive frames generally involve non-integer coordinates in at least one of them. Reprojection strategies must therefore *resample* any data that is moved between frames. Bilinear filtering is, by far, the most commonly used resampling strategy in real-time reprojection applications. Mappings between consecutive real-time frames tend to exclude large minifications, making trilinear filtering unnecessary. It is therefore important to understand the impact of bilinear filtering on the quality of reprojected data.

Although analyzing the effect of general motion across multiple frames is impractical, the special case of constant panning motion is easy to describe mathematically, particularly in one dimension (recall other types of motion can be approximated by translation, at least locally).

Assume we have information stored in a frame f_t that we want to resample to time $t+1$. Constant panning motion with velocity v can be described by $\pi_{t+1}(p) = p - v$, for every point p and time t. Without loss of generality, assume the velocity is in $[-0.5, 0.5]$. The entire resampling operation can be rephrased in terms of the discrete convolution

$$f_{t\to t+1} = f_t * \left[v \;\; (1\text{-}v)\right] \tag{7}$$
$$= f_t * k_v, \tag{8}$$

where we used the notation $f_{t\to t+1}$ to represent the new frame containing only reprojected data. Under our assumptions, the behavior of reprojection is therefore controlled by the effect of the convolution kernel $k_v = [v \;\; (1\text{-}v)]$.

For each different velocity v, and for each frequency ω, we compute the amplitude attenuation and the phase error introduced by k_v. Resulting plots are shown in Figure 9, where shaded regions represent values between the extremes. As we can see from the plots, reprojection through bilinear resampling tends to attenuate and misplace high-frequencies. Not visible from the plot is the fact that the problem is particularly extreme when $v = \pm 0.5$ and that it disappears when $v = 0$ (as expected from the interpolation property).

The effect of repeated resampling can also be analyzed:

$$f_{t\to t+n} = f_{t\to t+n\text{-}1} * k_v \tag{9}$$
$$= f_t * \overbrace{(k_v * \cdots * k_v)}^{n \text{ in total}}. \tag{10}$$

The trick is to interpret k_v as the probability mass function

© The Eurographics Association 2011.

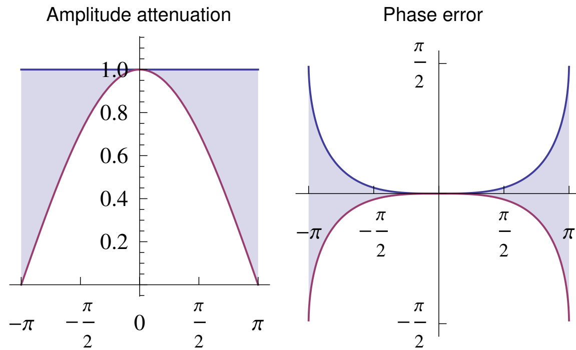

Figure 9: *Amplitude response and phase error associated to translation by linear resampling. Note that largest amplitude attenuation and phase error happens for high frequencies.*

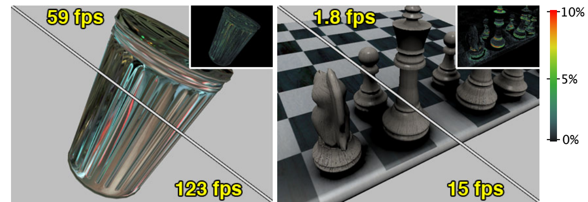

Figure 10: *Additional examples of shading acceleration using RRC. Each image compares (top) an input pixel shader to (bottom) a version modified to cache some partial shading computations over consecutive frames. The shading error after applying the cache is illustrated in the inset images.*

of a Bernoulli distribution with success probability v. The distribution has a variance of $\sigma^2 = v(1-v)$. Repeatedly convolving k_v with itself amounts to computing the sum distribution. By the Central Limit Theorem, this quickly converges to a Gaussian. By the sum property of variance, we have $\sigma_n^2 = nv(1-v)$. The progressively low-pass nature of repeated resampling then becomes obvious in the formula for the variance.

There are several alternatives to prevent the excessive blur introduced by repeated resampling from causing objectionable rendering artefacts. For example, we can periodically recompute values instead of relying on reprojection. This is in fact the approach followed by Sitthi-amorn et al. [SaLY*08b] (Section 3.4). Another alternative is to replace bilinear resampling with an alternative strategy that has better frequency properties, such as the one proposed by Yang et al. [YNS*09] (Section 5.3). Finally, in the context of computation amortization described in Section 3.5, we can also progressively attenuate the contribution of older frames, thereby limiting the maximum amount of visible blur.

5 Applications

5.1 Pixel shader acceleration

One of the direct uses of the reverse reprojection cache is to accelerate expensive pixel shading computations [NSL*07, SaLY*08a, SaLY*08b]. The basic idea is to bypass part or all computation of the original pixel shader whenever there are previous shading results available in the cache, as described in Section 3.4. Figure 4 shows the flow chart of this type of shading acceleration.

In addition to the marble shader described in Section 4.1, we show two more results of accelerating expensive pixel shaders using the RRC [SaLY*08b]. The first shader is a *Trashcan* environmental reflection shader from ATI's *Toyshop* demo, which combines a simple base geometry with a high-resolution normal map and environment map to reproduce the appearance of a shiny trashcan. The shader combines 25 stratified samples of an environment map using a Gaussian kernel to attenuate aliasing artifacts. In this example, we found that caching the sum of 24 samples of the possible 25 gives the most effective speed up without introducing too much visible artifacts (see Figure 10 (left) for a comparison). In other words, the modified shader evaluates 24 samples every fourth frame (on average) and evaluates the single sample with the greatest reconstruction weight at every frame. Indeed, this shader is not particularly suited for using TC to accelerate, because all of the calculations depend strongly on the camera position and cached values quickly become stale. Nevertheless, RRC provides a 2.1× performance improvement at an acceptable level of error.

The second shader computes an approximate object space ambient occlusion at each pixel for a chessboard scene with the king piece moving and the rest pieces static. The basic idea is to approximate the scene geometry as a collection of discs which are organized in a hierarchical data structure and stored as a texture. As each pixel is shaded, this data structure is traversed to compute the percentage of the hemisphere that is occluded. This calculation is combined with a diffuse texture and a Blinn-Phong specular layer to produce the final color. In this particular scene, the ambient occlusion calculation is carried out by summing the contribution of the king chess piece separately from the other pieces. We found that caching the portion of the ambient occlusion calculation that accounts for only the static pieces gives the best result. In other words, the contribution of the moving king and the remaining shading are recomputed at every frame. This provides a 8× speed-up for a marginal level of error and is demonstrated in Figure 10 (right). Caching more computations such as the entire ambient occlusion calculation will lead to visible error in the result although the speed-up factor is also larger (15× or more).

5.2 Multi-pass effects

Effects such as motion blur and depth-of-field are most easily understood and implemented as the accumulation of a series of frames, respectively rendered under slight variations in animation time or camera position, relative to a central

© The Eurographics Association 2011.

frame [HA90]. Although rendering and accumulating multiple frames in order to produce a single output frame may seem prohibitively expensive, the small magnitude of variation in input parameters between each accumulated frame leads to large amounts of coherence between them. This coherence has been successfully exploited in the context of image-based rendering [CW93], ray-traced animation sequences [HDMS03], and more recently in real-time rendering [NSL*07, YWY10]. Imperfections tend to be hidden by the low-pass nature of these effects, leading to images that are virtually indistinguishable from the brute-force results. The savings in rendering cost can be used to either increase quality by raising the number of accumulated frames, or to increase the frame rate for a fixed quality setting.

The real-time approach proposed in [NSL*07] starts by completely rendering a central frame into a buffer. Then, when rendering the accumulated frames, shading information is obtained from the central frame by reverse reprojection. The extent to which performance is improved depends on the relative cost between rendering the central frame (geometry + shading) and rendering each accumulated frame (geometry + cache-lookup). This is because reverse reprojection requires rasterizing the geometry of each accumulated frame (see Section 3.1). Improvements are therefore limited when geometry is complex and shading is relatively simple. Yu et al. [YWY10] propose to use forward reprojection (Section 3.2) in order to decouple this overhead from geometry complexity. They also apply a blurring pass to the reprojected frames before accumulation, so that the undersampling and disocclusion artifacts are attenuated.

Another rendering scenario that is closely related to depth-of-field is stereographic rendering. Two views are rendered from the same scene, one from the viewpoint of each eye of a virtual observer. Then, one of many different methods is used to expose each of the user's eyes to the corresponding image (e.g. shutter glasses, polarization filters), leading to the perception of depth. Stereographic rendering has recently gained increased attention given the success of 3D cinematographic productions as well as the increased availability of 3D-capable consumer hardware (TV sets, portable videogame consoles etc).

One way to avoid the doubling of cost-per-frame that would result from the brute-force approach to stereographic rendering is to instead render only one frame from the stereo pair and then warp it to produce the other frame. This is a well established idea that was successfully used in the context of stereographic ray-tracing [AH93] (where rendering cost was extremely high) and in stereographic head-tracked displays [MB95] (where warping was used to efficiently update a previously rendered stereo pair to compensate for user head movements).

Since per-pixel depth information is a natural by-product of real-time rendering, generating the mapping between two stereo views is particularly easy. The challenges are in the design of an efficient warping procedure that adapts to sharp features and attenuates any artefact resulting from surface points that are only visible from one of the viewpoints.

One way to perform this operation is to rely on an adaptive warping grid [DRE*10] (see Section 3.2) to transform one view into another. Didyk et al. further propose to exploit temporal coherence, by analyzing the camera movement from one frame to the next. Depending on the camera movement and the previously computed frame, it can be more advantageous to render and then warp either the left or the right eye view. E.g., imagine a panning motion from left to right. Here, a right eye view in frame i might be very close to a left eye view in frame $i+1$. Consequently, it makes sense to render the right eye view in frame $i+1$. The rendered frame and the previous are then warped to produce a left eye view for frame $i+1$. In particular, for a static camera and scene, the result is indistinguishable from a two-view rendering.

5.3 Shading antialiasing

One of the direct applications of amortized sampling (Section 3.5) is to supersample procedural shading effects, which usually contain high-frequency components that are prone to aliasing artifacts. By accumulating jittered samples generated in previous frames using amortized sampling, the extra frequency bands can be effectively suppressed. However, supersampling usually requires a small exponential smoothing factor α in order to gather sufficient samples. This has an undesired side effect that the running estimate can be overblurred because of excessive repeated resampling of the cache (Section 4.2).

Yang et al. [YNS*09] propose to keep a higher-resolution (2×2) running average in order to counteract this overblurring artifact. To reduce the overhead of maintaining such a high-resolution buffer, they store the 2×2 quadrant samples of each pixel into four subpixel buffers $\{b_k\}, k \in \{0,1,2,3\}$ using the interleaved sampling scheme. Each subpixel buffer is screen sized and manages one quadrant of a pixel. These subpixel buffers are updated in a round-robin fashion, i.e. only one per frame.

Reconstructing a subpixel value from the four subpixel buffers involves more work. Note that in the absence of scene motion, these four subpixel buffers effectively form a higher-resolution framebuffer. However, under scene motion, the subpixel samples computed in earlier frames reproject to offset locations. Conceptually, Yang et al. [YNS*09] forward reproject all the previous samples into the current frame and compute a weighted sum of these samples using a tent kernel, as indicated in Figure 11. This effectively reduces the contribution of distant samples and limits the amount of blur introduced. It also correctly handles both static and moving scenes simultaneously.

In addition to the higher resolution buffer, they also propose empirical methods to estimate reconstruction errors as well as the amount of signal change in real-time, and limit α

© The Eurographics Association 2011.

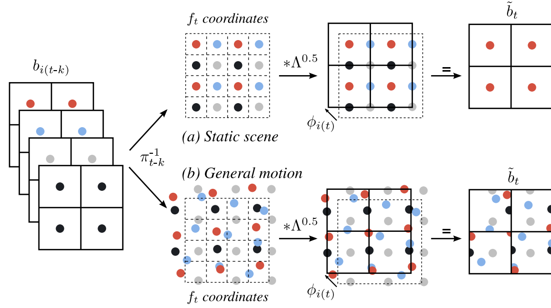

Figure 11: *Sampling from multiple subpixel buffers. To properly reconstruct the quadrant value, Yang et al. [YNS*09] use nonuniform blending weights defined by a tent function centered on the quadrant being updated. (a) In the absence of local motion, only the correct pixel has non-zero weight in the tent, so no resampling blur is introduced; (b) For a moving scene, the samples are weighted using the tent function, and higher weights are given to samples closer to the desired quadrant center to limit the amount of blur.*

accordingly such that a minimum amount of refresh is guaranteed. The reconstruction error is estimated by deriving an empirical relationship between the fractional pixel velocity **v**, α, and the error. Signal change, on the other hand, is estimated by a smoothed residual between the aliased sample and the history value. The user set thresholds for both errors, and the bounds for α are computed based on the error values.

Figure 12 shows the result of applying amortized sampling to antialiasing a horse-checkboard scene, which includes an animated wooden horse galloping over a marble checkered floor. The result using 4×4 subpixel buffers shows significant improvement over regular amortized sampling ($1\times$ viewport-sized cache), with only a minor sacrifice of speed. In fact, the PSNR shows that this technique offers better quality when compared to the conventional 4×4 stratified supersampling, which runs at a six times lower framerate.

5.4 Shadows

Shadows are widely acknowledged to be one of the global lighting effects with the most impact on scene perception. They are perceived as a natural part of a scene and give important cues about the spatial relationship of objects.

Due to its speed and versatility, shadow mapping is one of the most used real-time shadowing approaches. The idea is to first create a depth image of the scene from the point of view of the light source (shadow map). This image encodes the front between lit and unlit parts of the scene. On rendering the scene from the point of view of the camera each fragment is transformed into this space. Here the depth of each transformed camera fragment is compared to the respective depth in the shadow map. If the depth of the camera fragment is nearer it is lit otherwise it is in shadow (see Figure 13).

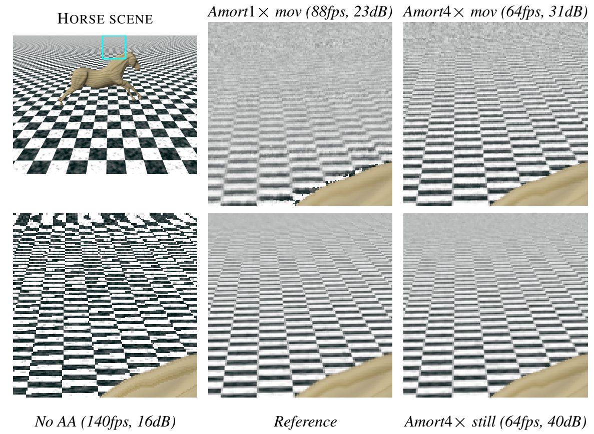

Figure 12: *Comparison between no antialiasing, amortized supersampling with viewport-size cache (Amort$1\times$), amortized supersampling with improved 2×2 subpixel buffers (Amort$4\times$), and the ground-truth reference result for a horse-checkerboard scene [YNS*09]. The $4\times$ still image approaches the quality of the reference result, whereas the motion result provides an acceptable approximation without overblurring.*

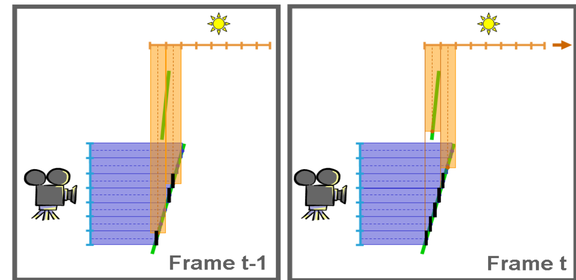

Figure 13: *If the rasterization of the shadow map changes (here represented by a right shift), the shadowing results may also change. On the* left *three fragments are in shadow, while on the* right *five fragments are in shadow. This results in flickering or swimming artifacts in animations.*

5.4.1 Pixel correct shadows

The most concerning visual artifacts of shadow mapping originate from aliasing due to undersampling. The cause for undersampling is in turn closely related to rasterization that is used to create the shadow map itself. Rasterization uses regular grid sampling for rasterization of its primitives. Each fragment is centered on one of these samples, but is only correct exactly at its center. If the viewpoint changes from one frame to the next, the regular grid sampling of the new frame is likely to be completely different than the previous one. This frequently results in artifacts, especially noticeable

© The Eurographics Association 2011.

for thin geometry and the undersampled portions of the scene called *temporal aliasing*.

This is especially true for shadow maps. Due to shadow map focusing, a change in the viewpoint from one frame to the next also changes the regular grid sampling of the shadow map. Additionally the rasterized information is not accessed in the original light-space where it was created, but in eye-space, which worsens these artifacts. This frequently results in temporal aliasing artifacts, mainly flickering (See Figure 13).

The main idea in [SJW07] is to jitter the view port of the shadow map differently in each frame and to combine the results over several frames, leading to a higher effective resolution. Figure 15 shows the gradual refinement after accumulating results from multiple frames.

Exponential smoothing as described in Section 3.5 is employed here on the shadow map tests $s_t[\mathbf{p}]$. This serves a dual purpose. On the one hand temporal aliasing can be reduced by using a small smoothing factor α. On the other hand, the shadow quality can actually be made to converge to a pixel-perfect result by optimizing the choice of the smoothing factor.

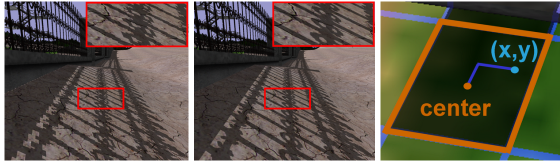

Figure 14: *LiSPSM* (left) *gives good results for a shadow map resolution of* 1024^2 *and a view port of* 1680×1050, *but temporal reprojection* (middle) *can still give superior results because it uses shadow test confidence, defined by the maximum norm of shadow map texel center and current pixel* (right).

The smoothing factor α allows balancing fast adaption on changing input parameters against temporal noise. With a larger smoothing factor, the result depends more on the new shadow results from the current frame and less on older frames and vice versa. To this end, the smoothing factor is determined per-pixel according to the *confidence* of the shadow lookup. This confidence is defined to be higher if the lookup falls near the center of a shadow map texel, since only near the center of shadow map texels it is very likely that the sample actually represents the scene geometry (see Figure 14). In the paper the maximum norm of the current pixel $\mathbf{p}$ and the shadow map texel center $\mathbf{c}$ is used to account for this

$$\mathsf{conf} = (1 - \max(|\mathbf{p}_x - \mathbf{c}_x|, |\mathbf{p}_y - \mathbf{c}_y|) \cdot 2)^m, \tag{11}$$

but other norms could be used as well. The parameter m defines how strict this confidence is applied. $m < 4$ results in fast updates were most shadow map lookups of the current frame have a big weight and the resulting shadow has noisy edges. $m > 12$ results in accurate but slow updates were most lookups from the current frame have small weight.

The authors found out that m should be balanced with camera movement. When the camera moves fast m can be small because noise at the shadow borders is not noticed. Only for a slowly moving camera or a still image are higher values of m necessary. This is motivated by the human visual system, which tends to integrate over motion, thereby allowing for noisier edges when strong movement is present. This confidence can now be directly used in the exponential smoothing formula (see Section 3.5)

$$f_t[\mathbf{p}] \leftarrow (\mathsf{conf}) s_t[\mathbf{p}] + (1 - \mathsf{conf}) f_{t\text{-}1}(\pi_{t\text{-}1}(\mathbf{p})). \tag{12}$$

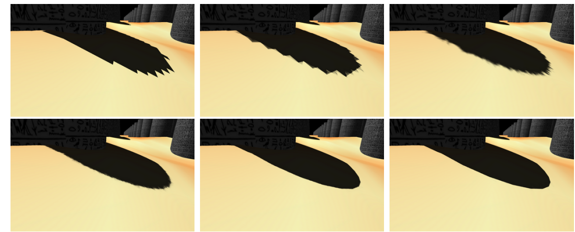

Figure 15: *Shadow adaption over time of an undersampled uniform shadow map after 0* (top-left), *1* (top-middle), *10* (top-right), *20* (bottom-left), *30* (bottom-middle) *and 60* (bottom-right) *frames.*

5.4.2 Soft shadows

In reality most light sources are area light sources and hence most shadows exhibit soft borders. *Light source sampling* [HH97] creates a shadow map for every sample (each on a different position on the light source) and calculates the average (= soft shadow) of the shadow map test results s_i for each pixel (see Figure 16). Therefore, the soft shadow result from n shadow maps for a given pixel $\mathbf{p}$ can be calculated by

$$\psi_n(\mathbf{p}) = \frac{1}{n} \sum_{i=1}^{n} s_i(\mathbf{p}). \tag{13}$$

The primary problem here is that the number of samples (and therefore shadow maps) to produce smooth penumbrae is huge. Therefore this approach can be inefficient in practice. Typical methods for real-time applications approximate an area light by a point light located at its center and use heuristics to estimate penumbrae, which leads to soft shadows that are not physically correct (see Figure 17, left). Here overlapping occluders can lead to unnatural looking shadow edges, or large penumbrae can cause single sample soft shadow approaches to either break down or become very slow.

One observation is that the shadow sampling can be extended over time. It is for example possible to change the sampling pattern on the source in each frame, hereby trading aliasing artifacts with less objectionable random

© The Eurographics Association 2011.

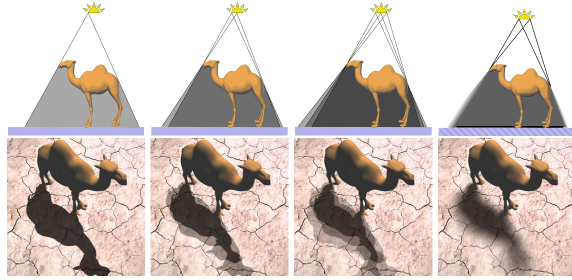

Figure 16: *Light sampling with 1, 2, 3 and 256 shadow maps* (left *to* right).

noise. This is particularly easy to achieve for symmetric light sources [ED07b, SEA08]. More generally, light source area sampling can be formulated in an iterative manner [SSMW09], by evaluating only a single shadow map per frame. Reformulating Equation 13 gives

$$\psi(\mathbf{p}) = \frac{s(\mathbf{p}) + \Sigma(\mathbf{p})}{n(\mathbf{p}) + 1} \qquad \Sigma(\mathbf{p}) = \sum_{i=1}^{n(\mathbf{p})} s_i(\mathbf{p}). \tag{14}$$

were $s(\mathbf{p})$ is the hard shadow map result for the current frame and pixel and $n(\mathbf{p})$ is the number of shadow maps evaluated until the last frame for this pixel. Note that now n depends on the current pixel because dependent on how long this pixel has been visible, a different number of shadow maps may have been evaluated for this pixel. Calculation of this formula is straight forward if $n(\mathbf{p})$ and $\Sigma(\mathbf{p})$ are stored in a buffer (another instance of the RRC: see Section 3.1). With this approach, the soft shadow improves from frame to frame and converges to the true soft shadow result if pixels stay visible "long enough" (see Figure 18, upper row).

In practice this can result in temporal aliasing for small n.

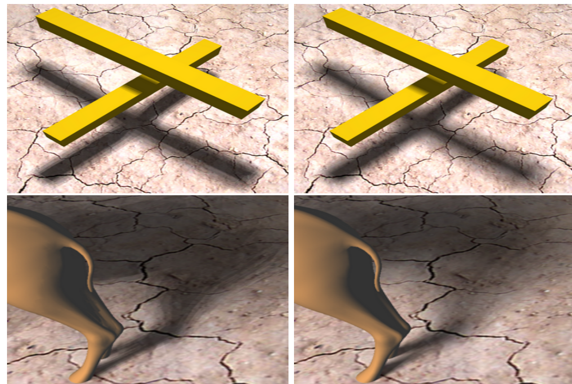

Figure 17: Left side*: PCSS 16/16; Overlapping occluders* (upper row) *and bands in big penumbras* (lower row) *are known problematic cases for single-sample approaches.* Right side*: soft shadows exploiting TC*

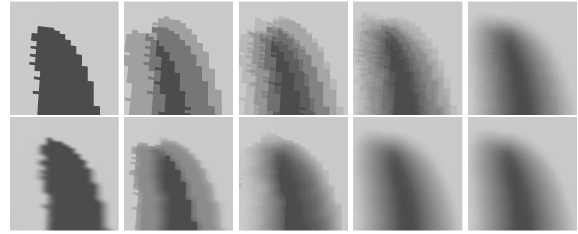

Figure 18: *Convergence after 1,3,7,20 and 256 frames;* upper row*: sampling of the light source one sample per frame;* lower row*: soft shadows with TC.*

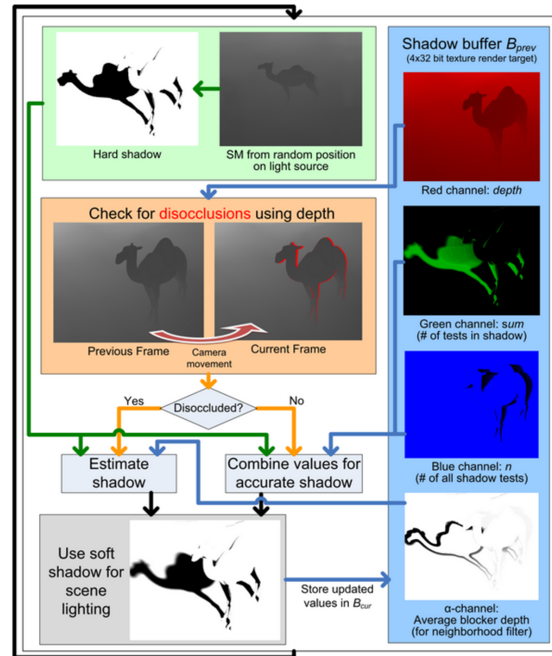

Figure 19: *Structure of the soft shadows with TC algorithm.*

Care has to be taken how to manage those cases. When a pixel becomes newly visible and therefore no previous information is available in the RRC, a fast single sample approach (PCSS with a fixed 4x4 kernel) is employed to generate an initial soft shadow estimation for this pixel. For all other n the expected standard error is calculated and if it is above a certain threshold (expected fluctuation in the soft shadow result in consecutive frames) a depth-aware spatial filter is employed to take information from the neighborhood in the RRC into account (see Figure 19). This approach largely avoids temporal aliasing and can be nearly as fast as hard shadow mapping if all pixels have been visible for some time and the expected standard error is small enough (see Figures 18 and 17).

© The Eurographics Association 2011.

5.5 Global Illumination

It is a major goal of real-time research to achieve plausible (and in the long run, physically correct) global illumination. In this section, we present several techniques that explore TC in the attempt to approximate global illumination effects in real-time. Many techniques can be found in the excellent survey by Damez et al. [DDM03]. Nonetheless, the focus is often on off-line solutions or it is assumed that knowledge of subsequent keyframes is available. For interactive rendering this is not always achievable and it is difficult to exploit such algorithms on current GPUs that are in the focus of our overview.

The radiance emitted from point p into direction ω can be described by the rendering equation [Kaj86, ATS94]

$$L(\mathbf{p},\omega) = L_e(\mathbf{p},\omega) + \frac{1}{\pi}\int_{\Omega} f_r(\mathbf{p},\omega',\omega)L_i(p,\omega')(\mathbf{n_p}\cdot\omega')d\omega'. \quad (15)$$

Ω denotes the space of all hemispherical directions, L_e is the self emission, f_r is the *bidirectional reflectance distribution function* (BRDF), L_i is the incident light from direction ω', and n_p is the surface normal.

Global illumination algorithms often use Monte-Carlo sampling to evaluate this multi-dimensional integral in a feasible way. We can exploit TC between consecute frames, e.g., by spreading the evaluation of the integral over time.

5.5.1 Screen-space ambient occlusion

Ambient occlusion [CT81] is a cheap but effective approximation of global illumination which shades a pixel with the percentage of the hemisphere that is blocked. It can be seen as the diffuse illumination of the sky [Lan02]. Ambient occlusion of a surface point $\mathbf{p}$ is computed as

$$\mathrm{AO}(\mathbf{p},\mathbf{n_p}) = \frac{1}{\pi}\int_{\Omega} V(\mathbf{p},\omega')\left(\mathbf{n_p}\cdot\omega'\right)\, d\omega'. \quad (16)$$

V is the inverse binary visibility function, with $V(\mathbf{p},\omega') = 1$ if the visibility in this direction is blocked by an obstacle, 0 otherwise.

Screen-space ambient occlusion (SSAO) methods [Mit07] sample the frame buffer as a discretization of the scene geometry. These methods are of particular interest for real-time applications due to the fact that the shading overhead is mostly independent of scene complexity, and several variants of SSAO have been proposed since [FC08, BSD08, SKUT*10]. We assume that any SSAO method can be written as an average over contributions C depending on a series of samples $\mathbf{s_i}$:

$$SSAO_n(\mathbf{p}) = \frac{1}{n}\sum_{i=1}^{n} C(\mathbf{p},\mathbf{s_i}), \quad (17)$$

where a typical contribution function for a single SSAO sample can be

$$C(\mathbf{p},\mathbf{s_i}) = V(\mathbf{p},\mathbf{s_i})\max(\cos(\mathbf{s_i}-\mathbf{p},\mathbf{n_p}),0). \quad (18)$$

$\mathbf{s_i}$ is an actual sample point around $\mathbf{p}$, and $V(\mathbf{p},\mathbf{s_i})$ is now a binary visibility function that is resolved by evaluating the depth test for $\mathbf{s_i}$.

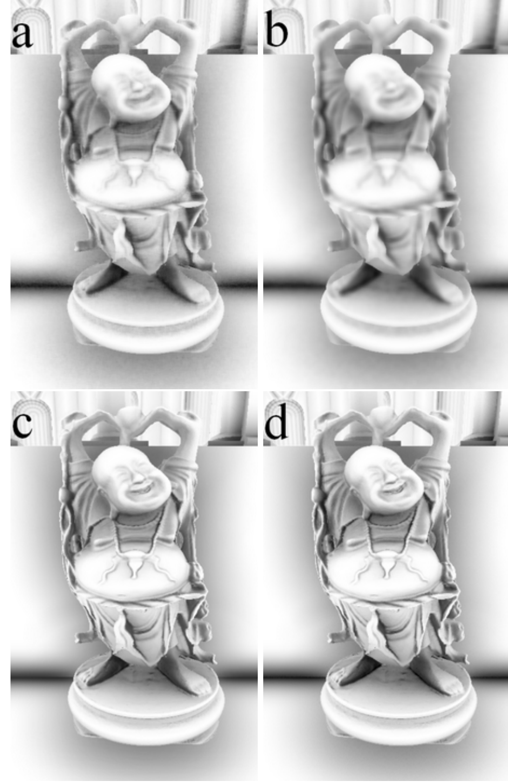

Figure 20: *SSAO without TC using 32 samples per pixel with (a) a weak blur, (b) a strong blur (both 23 FPS), (c) temporal SSAO using 8–32 samples (initially 32, 8 in a converged state) (45 FPS). (d) Reference solution using 480 samples (2.5 FPS). The scene has 7M vertices and runs at 62 FPS without SSAO.*

Reverse reprojection allows us to cache and reuse previously computed SSAO samples. The properties of SSAO (relatively low-frequency, independence from light-source, local support of the sampling kernel) are beneficial for using TC, as it was already demonstrated in commercial games [SW09]. In the following we discuss the temporal SSAO (TSSAO) method of Mattausch et al. [MSW10], who focus on improving the accuracy and visual quality of SSAO for an equal number of samples per frame or less, and introduce an invalidation scheme that handles moving objects well.

A comparison of conventional SSAO with TSSAO is shown in Figure 20. The noisy appearance of a coarse SSAO solution that uses only a few samples (image a) can be improved with a screen-space spatial discontinuity filter. However, the result of this operation can be quite blurry (image b). As long as there is a sufficient history for a pixel, TSSAO produces smooth but crisp SSAO without depending on heavy post-processing (image c).

© The Eurographics Association 2011.

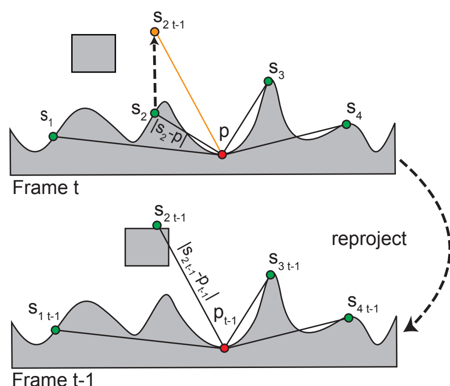

Figure 21: *The distance of* $\mathbf{p}$ *to sample point* $\mathbf{s_2}$ *in the current frame differs significantly from the distance of* $\mathbf{p_{t-1}}$ *to* $\mathbf{s_{2_{t-1}}}$ *in the previous frame, hence we assume that a local change of geometry occurred, which affects the shading of* $\mathbf{p}$.

Integration over time: In frame t, we calculate a new contribution C_t from k new SSAO samples.

$$C_t(\mathbf{p}) = \frac{1}{k} \sum_{i=j_t(\mathbf{p})+1}^{j_t(\mathbf{p})+k} C(\mathbf{p}, \mathbf{s_i}), \tag{19}$$

where $j_f(\mathbf{p})$ counts the number of unique samples that have already been used in this solution. We combine the new contribution with the previously computed solution

$$SSAO_t(\mathbf{p}) = \frac{w_{t-1}(\mathbf{p}_{t-1}) SSAO_{t-1}(\mathbf{p}_{t-1}) + kC_t(\mathbf{p})}{w_{t-1}(\mathbf{p}) + k} \tag{20}$$

$$w_t(\mathbf{p}) = min(w_{t-1}(\mathbf{p}_{t-1}) + k, w_{max}). \tag{21}$$

The weight w_{t-1} represents the number of samples that have already been accumulated in the solution, until w_{max} has been reached. The solution converges very quickly, and this predefined maximum controls the refresh rate, and ensures that the influence of older contributions decays over time.

Note that for TSSAO, spatial filtering only has to be applied in regions where the solution has not sufficiently converged. This is done by shrinking the screen-space filter support proportionally to the convergence w_n/w_{max}. The results of the filtering can be further improved by making it *convergence-aware*, i.e., assigning higher weights to sufficiently converged filter samples.

Detecting changes: Special attention must be paid to the detection of cache misses (i.e., pixels with an invalid SSAO solution). A cached value of a pixel is invalid if either one of the following three conditions has occurred: 1) a disocclusion of the current pixel, 2) the pixel was previously outside the frame buffer. 3) a change in the *sample neighborhood* of the pixel. Case 1) and case 2) can be handled like conventional cache misses as described previously in Section 3.3. However, we additionally have to check for case 3), because nearby changes in the geometry can affect the shading of the current pixel.

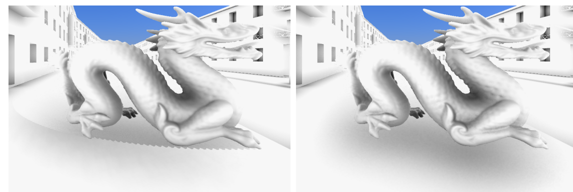

Figure 22: *Moving dragon model using no invalidation (left, causing severe artifacts in the shadow), and using an invalidation factor set to a proper value (right, no artifacts).*

Checking the complete neighborhood of a pixel would be prohibitively expensive, and therefore we use sampling (i.e., we simply reuse the available set of samples generated for computing $C_t(p)$). We can estimate the validity of reusing sample position $\mathbf{s_{i_{t-1}}}$ in the current frame by evaluating a measure of change with respect to the configuration of $\mathbf{p}$ *relative* to $\mathbf{s_i}$ between frame $t-1$ and t. As illustrated in Figure 21, we compute the distance differences

$$\delta(s_i) = ||\mathbf{s_i} - \mathbf{p}| - |\mathbf{s_{i_{t-1}}} - \mathbf{p_{t-1}}||. \tag{22}$$

as our measure of change. Note that we could have additionally used the angular differences (i.e., $|\cos(\mathbf{s_i} - \mathbf{p}, \mathbf{n_p}) - \cos(\mathbf{s_{i_{t-1}}} - \mathbf{p_{t-1}}, \mathbf{n_{p_{t-1}}})|$). In this case however, we would have to store the surface normals of every pixel in the previous frame. It is sufficient to use those samples that lie in front of the tangent plane of $\mathbf{p}$ for the neighborhood test, since only those samples actually modify the shading.

Smooth invalidation: Consider for example a slowly deforming surface, where the SSAO will also change slowly. In such a case it is not necessary to fully discard the previous solution. Instead we introduce a new continuous definition of invalidation that takes a measure of change into account. This measure of change is given by $\delta(\mathbf{s_i})$ at validation sample position $\mathbf{s_i}$, as defined in Equation 22. In particular, we compute a *confidence* value $\mathsf{conf}(\mathbf{s_i})$ between 0 and 1. It expresses the degree to which the previous SSAO solution is still valid:

$$\mathsf{conf}(\mathbf{s_i}) = 1 - \frac{1}{1 + S\delta(\mathbf{s_i})}. \tag{23}$$

The invalidation factor S is a parameter which controls the smoothness of the invalidation. The overall confidence $\mathsf{conf}(\mathbf{p})$ in the previous SSAO solution is given by $\min(\mathsf{conf}(\mathbf{s_0}), .., \mathsf{conf}(\mathbf{s_k}))$. This value is used to attenuate the weight w_t given to the solution of the previous frame in Equation 21.

Figure 22 shows the effect of the invalidation and smooth invalidation factor on a scene with a moving object. Setting the invalidation factor S to $(15 \leq S \leq 30)$ usually gives good results.

© The Eurographics Association 2011.

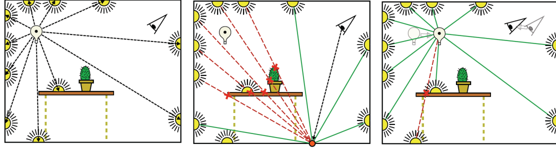

Figure 23: *(Left) Instant radiosity shoots paths from the light source, and creates virtual point lights at the intersection with geometry. (Middle) Using shadow maps, the visibility of each VPL and their contribution to the current image is determined. (Right) Temporal coherence: When the view point or light source moves, one of the VPLs becomes invisible from the light source, all the others are reused. Image courtesy of Samuli Laine.*

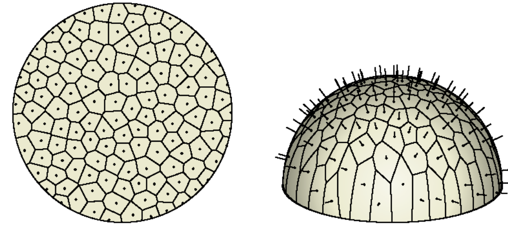

Figure 24: *An uniform distribution on the unit disc corresponds to a cosine-weighted distribution on a hemisphere. The VPL management aims to keep the uniformity of the VPLs on the unit disc while recomputing a budget of VPLs per frame. Image courtesy of Samuli Laine.*

5.5.2 Instant radiosity

Instant radiosity [Kel97] is a hardware-friendly global illumination method that computes so called virtual point lights (VPLs) along the intersections of a light path with a surface, and uses them for indirect scene illumination. The visibility is resolved by computing an individual shadow map for each VPL. The shadow map computation is also the main bottle neck of the algorithm, as it requires to sample the scene many times for a reasonable number of VPLs. This drawback prevents real-time frame rates for the original version of this algorithm. In the following we will demonstrate how to use object-level and pixel-level TC to improve performance and visual quality of this important global illumination algorithm.

Incremental instant radiosity: By reusing VPL visibility over time, Laine et al. [LSK*07] exploit object-level TC to reach real-time frame rates. For the sake of performance, this algorithm only computes first-bounce indirect illumination, which is sufficient in most cases. Nevertheless, hundreds of shadow maps are needed for convincing global illumination. In order to keep the number of VPL computations per frame feasible for real-time purposes, this algorithms reuses the valid VPLs from the last frame and recomputes only a *small budget* of invalid shadow maps in a frame. A VPL stays valid if it is within the light frustum and is not occluded from the light source (which is tested with a ray caster). The algorithm is visualized in Figure 23.

The main task of this algorithm is to incrementally keep a good distribution of the VPLs during consecutive frames. Assuming a 180° spotlight, we use the fact that a cosine-weighted distribution on a hemisphere corresponds to a uniform distribution on a disc (shown in Figure 24). In order to manage the VPL distribution on the unit disc, the algorithm creates a 2D Delaunay triangulation. To choose the best position for new VPLs, we minimize *dispersion*, which is computed as the radius of the largest empty circle that contains no sample points. In case of omni-directional light sources, we operate on the unit sphere instead of the unit disc.

Note that the algorithm captures changes in the scene with a certain latency, and shadows cast from dynamic objects are not supported. The authors report a speedup from 1.4–6.8 for different scenes and resolutions. In their tests, they fixed the number of VPLS to 256 and the recomputation budget to 4–8 VPLs.

Imperfect shadow maps: Based on the observation that coarse visibility is sufficient for low-frequency global illumination, Ritschel et al. [RGK*08] significantly accelerate the VPL generation for instant radiosity. They use a point-based scene representation, and distribute these points among the VPLs to generate so called imperfect shadow maps. While each shadow map is sampled with only a coarse subset of scene points, holes can be closed with a push-pull algorithm. This method allows hundreds of shadow map-based visibility queries per frame in at least interactive time.

However, even such a large number of queries are insufficient to avoid typical undersampling artifacts, e.g., resulting in flickering between frames if the VPLs are recomputed. In order to improve the visual quality and reduce these artifacts, it is straightforward to combine the imperfect shadow mapping approach with temporal reprojection.

The main problem of using TC for global illumination is the global nature of changes of the lighting conditions and the scene configuration – some tradeoff between smoothing and correctness is inevitable and a satisfactory general solution is hard to find. Knecht et al. [KTM*10] chose to use a confidence value instead of a binary threshold for invalidation. In particular, the confidence in reusing a previous solution is guided by the amount of change of a pixel between the previous and current frame. They introduce a couple of parameters of a rather ad-hoc nature:

$$\begin{aligned}\varepsilon_{pos} &= \|(x_t - x_{t-1}; y_t - y_{t-1}; d_t - d_{t-1})\mathbf{w_p}\| \\ \varepsilon_{norm} &= (1 - n \cdot n_{prev})w_n \\ \varepsilon_{ill} &= saturate(\|I_t - I_{t-1}\|^3)w_i \\ conf &= saturate(1 - \max(\varepsilon_{pos}; \varepsilon_{norm}; \varepsilon_{ill}))c_B. \quad (24)\end{aligned}$$

© The Eurographics Association 2011.

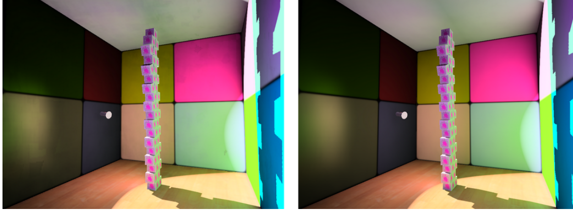

Figure 25: *Imperfect shadow maps still show some artifacts with 256 VPLS, which can be smoothed out using TC. Image courtesy of Martin Knecht.*

The first equation computes a distance value using screen-space position and depth, the second equation takes the differences in the normals into account, the third the difference in the illumination values. The weights $\mathbf{w_p}$, w_n, and w_i are highly scene dependent and require fine tuning of the user. The final confidence is computed as the maximum of these measures multiplied with some base confidence c_B, and is then used as the weight of a standard exponential smoothing operation (see Section 3.5).

As can be seen in Figure 25, TC improves the quality and reduces the noise (resulting in distracting flickering artifacts during animations). Due to the low frequency nature of indirect illumination, the motion blur like artifacts caused by moving light sources and animated objects are not very distracting in the general case.

5.6 Spatio-temporal upsampling

In addition to TC, spatial coherence may also exist within shading signals (e.g. low-frequency diffuse shading). Herzog et al. [HEMS10] propose a spatio-temporal upsampling technique that exploits temporal and spatial redundancy. Strong temporal changes (e.g., moving lights) are handled with spatial upsampling, while constancy is exploited to ensure a high-quality convergence via temporal upsampling. Such spatio-temporal filtering is often applied for video restoration [Tek95, BM05] and can also be used to suppress aliasing artifacts [Shi95].

The basic approach of spatio-temporal upsampling follows a joint- or cross- bilateral upsampling [TM98, SB95, ED04, PSA*04, KCLU07] scheme:

$$f_t(\mathbf{p}) = \frac{1}{\sum w^s w^t w^f} \sum_{q=0}^{T} \sum_{j \in \mathcal{N}\{\mathbf{p}_q\}} w^s(\mathbf{p}_q, j)\, w^t(\mathbf{p}_q, j)\, w^f(q)\ f^l_{t-q}(\hat{j}), \quad (25)$$

where $\mathcal{N}$ describes a spatial neighborhood around a pixel and q is an index that indicates the frames over time. Hence, the double summation takes space and time into account. w^s is a spatial weight that evaluates world-space distance and similarity of samples based on surface properties such as normals or material indices. w^t is a binary occlusion test (checking whether the reprojected pixel $\mathbf{p}_q = \pi_{t\text{-}q}(\mathbf{p})$ is actually visible in the corresponding frame). w^f describes a temporal fade-out that reduces the influence of older pixels. $\frac{1}{\sum w^s w^t w^f}$ is a normalization term that normalizes the weighting coefficients. f^l_{t-q} are low resolution frames that were created using an interleaved pixel refresh at time $t-q$ and consequently, $\hat{j}$ describes the corresponding position of the position j in the low resolution image. While, in theory, it seems that many previous frames have to be kept in memory, choosing exponential weights allows for an accumulation in a single history buffer [HEMS10].

Nonetheless, when involving samples from previous frames, it is important to detect which pixel shading values are still useful for the current frame. E.g., for a fast moving light, shadows might change their location and easily pollute a temporal integration. In order to capture these effects, Herzog et al. propose to examine the temporal gradient of the previously-constructed frame and the only spatially-upsampled current frame. If the gradient is low, more confidence is given to temporal weights, if not, the algorithm favors spatial upsampling techniques to produce the final high-quality version of the current frame. The intuition is simple. If a region of an image changed little over time, it is useful to exploit more samples from previous frames. Nonetheless, if strong changes occurred, older values should be considered unreliable. In order to make this solution more robust to outliers, a temporal smoothing is applied to the gradient.

Each low-resolution shading frame f^l_{t-q} is produced using interleaved sampling, meaning that the camera changed to ensure that when putting all low resolution images together, one can actually produce a complete high-resolution rendering without artifacts. In practice, the temporal fadeout makes it impossible to ensure a perfect match, but the quality is still higher than for spatial or temporal upsampling alone.

5.7 Frame interpolation

Frame interpolation is widely applied in video encoding and uses temporal redundancy to allow for a better compression behavior. We will investigate compression and streaming briefly in Section 5.10. Here, we analyze a second reason to employ frame-interpolation strategies: hold-type blur.

Nowadays, hold-type displays, such as LCD screens, show an image over a longer period of time instead of flashing it on the screen. The resulting perceptual effects are very interesting. In fact, moving content is perceived blurred because the eye tracks the content over the screen. During the eye motion, the image content stays partly constant (due to an insufficient framerate) which leads to an integration of the image on the retina (not unlike motion blur) [KV04]. For a long time, a lot of the blur perception was wrongly attributed to the display's response time, but Pan et al. [PFD05] showed that only 30 % of the perceived blur are a consequence of it. The remaining 70 % are mostly a result of hold-type blur. Especially for low frame rates this effect can have dramatic consequences and re-

© The Eurographics Association 2011.

duce the image quality drastically [Jan01], but even reaction times decrease and task performance is reduced [DER*10b].

Modern TVs try to optimize image quality by employing interpolation schemes [FPD08]. While an accurate interpolation and matching of content over time can become very difficult for a TV set because optical flow is challenging to compute robustly, a rendering context offers many advantages because the problem becomes actually much simpler. It is possible to derive accurate velocity and geometric information from a scene by simply rendering it into a buffer. Hereby, one can avoid approximate image-based estimates. TV sets are still successful in many cases because, at high frame rates, the precision of our perception is reduced. Consequently, intermediate images do not have to exhibit the same quality as key frames.

Didyk et al. [DER*10b] build upon the observation that intermediate frames can be of lower quality and exploit the effect algorithmically. They produce a high-frame rate sequence that is then directly fed unaltered to a high-refresh LCD screen.They rely on one key observation which is that the human visual system spreads high-frequencies of one frame over succeeding blurred frames if a sufficiently high frame rate is reached [TV05]. Hereby, they can hide potential artifacts in warped frames. Precisely, they extrapolate a given image and blur all parts of the warped image that might potentially exhibit a reduced image quality. The frequencies that are lost by the blurring process in the extrapolated frames can be compensated for in the unwarped original frame. In the affected regions, the amplitude of the high frequencies is increased according to the blur that is applied to the successive frames. Because artifacts are hidden by the blur, a very cost-effective grid warping strategy can be used. This grid is deformed by velocity vectors that are directly extracted from the scene and a snapping process ensures that the main discontinuities are respected. The technique is successful enough to enable the addition of two intermediate frames. In other words, a 40 Hz sequence can be transformed into a 120 Hz output that is almost indistinguishable from an actually-rendered 120 Hz sequence, which was confirmed by a user study.

Andreev [And10] also applies a temporal upsampling scheme, but makes use of a more costly warping strategy. They target only a single in-between frame to successfully transform 30 Hz sequences to 60 Hz. The idea is to rely again on a frame extrapolation, but to further separate static and dynamic content. Static elements are usually well handled by warping strategies, but dynamic objects can hide – and when warped, unveil – important parts of the scene. Consequently, holes can appear in the extrapolated frames. Andreev proposes to copy static pixel patches from the neighborhood to fill up these holes. The dynamic content is then added on top of the final shot. The algorithm is useful and well-adapted for current game consoles (XBox, PS3). It finds application in several shipping game titles which shows its practical

Figure 26: *Examples of stylizing a frame from a rendered 3D animation of a tank scene (left) and a 3D animation of a lizard with a still photograph in the background (right).*

relevance. Andreev [And10] also explored a solution that interpolates between two frames which gave more accurate results, but found that it had the drawbacks that it required more computational resources and added an extra frame of latency.

5.8 Non-photorealistic rendering

Real-time reprojection has also been used by a non-photorealistic rendering (NPR) system that converts animated scenes to artistic brush stroke renderings of different styles [LSF10]. Computing a new set of NPR strokes from scratch in each frame of an animation sequence results in significant flickering artifacts. Instead, the algorithm maintains temporal coherence by treating brush strokes as particles and advecting the vast majority of them accoding to the scene motion.

In order to advect brush strokes, the algorithm generates a buffer that stores per-pixel forward motion vectors for the animated scene. This buffer can be efficiently computed on the GPU by using reprojection to calculate the forward motion vector from frame $t-1$ to frame t in the vertex shader. The motion vector is interpolated by the hardware and provided as input to the pixel shader. Much like traditional reprojection, the pixel shader homogenizes the motion vector and then outputs the result to the render target in the clip space of frame $t-1$. Finally, each brush stroke particle uses this motion vector buffer to forward reproject its position from frame $t-1$ to frame t. Figure 26 shows examples of synthetic scenes rendered with this system.

5.9 Discrete LOD blending

The idea behind discrete *level-of-detail* (LOD) techniques is to use a set of representations with differing complexities (level-of-detail) for one model and select the most appropriate representation for rendering at runtime. Complexity can for instance vary in the employed materials or shaders or in the amount of triangles used. Due to memory constrains and the effort in creating them only a small number of LODs is being used and therefore switching from one representation to another can lead to noticeable popping artifacts. A theo-

© The Eurographics Association 2011.

retical solution would be to switch only when the respective pixel output of two representations is indistinguishable. This so called *late switching* has practical problems. First, it is hard to guarantee equality in pixel output for a given view scenario and lighting without rendering both representations first, which of course defeats the purpose. Second, the idea of switching as late as possible counteracts the potential gain of employing LODs in the first place. In practice switching is done as soon as "acceptable".

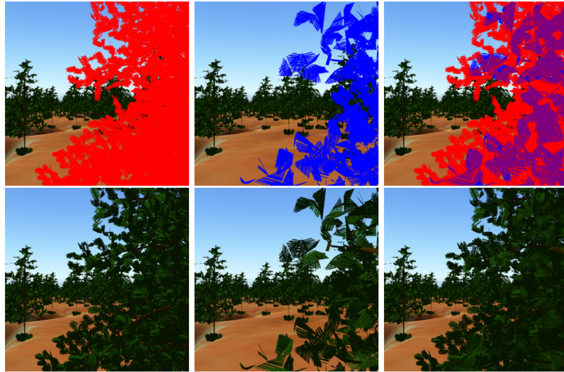

Figure 27: *LOD interpolation combines two buffers containing the discrete LODs to create smooth LOD transitions.* First and second column*: buffers;* last column*: combination. The top row shows the two LODs in red and blue respectively.*

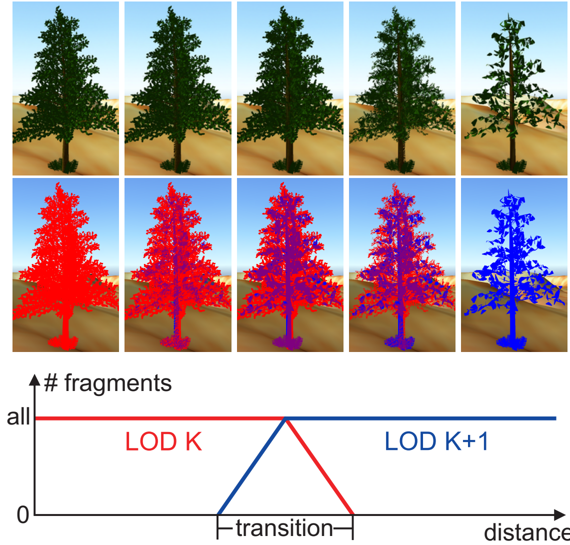

Figure 28: *Transition phase from LOD_k to LOD_{k+1}:* left:*LOD_k;* middle: *midway in the transition all fragments of both LODs are drawn;* right: *LOD_{k+1};* Below: *First LOD_{k+1} is gradually introduced till all its fragments are drawn. Then LOD_k is gradually removed by rendering fewer and fewer fragments. The top two rows show the result of our method and a false color illustration.*

A more practical solution to this problem proposed by [GW06] is to include a transition phase during which both LODs are rendered and then blended into the final image [GW06]. Apart from other problems, this approach requires that the geometry (and the shaders) of both LODs have to be rendered in this transition phase, thereby generating a higher rendering cost than the higher quality level alone would incur. To circumvent this [SW08] have introduced LOD interpolation (see Figure 27). The idea is that by using TC the two LODs required during an LOD transition can be rendered in *subsequent frames*. Two separate render passes are used to achieve the transition phase between adjacent LOD representations: Pass one renders the scene into an off-screen buffer (called *LOD buffer*). For objects in transition one of the two LOD representations is used and only a certain amount of its fragments are rendered (see Figure 28), depending on where in the transition (i.e., how visible) this object currently is. This is later repeated in the next frame using the other LOD representation and rendering into a second *LOD buffer*. The second pass combines these two *LOD buffers* (one from the current and one from the previous frame) to create the desired smooth transition effect.

To determine the number of fragments to render for a given representation, so-called *visibility textures* are used. Each encodes a visibility threshold function $visTex(\mathbf{p}) \rightarrow [0..1]$ that maps the object-space coordinate (before any animation is applied) of a given fragment $\mathbf{p}$ to the fragments visibility threshold.

This allows individual fragments to be discarded by comparing the output of this function to an objects visibility threshold ι. ι encodes were in the transition phase a representation currently is and is given by the transition function depicted in Figure 28

Writing this process as a function gives $discard : R^3 \times [0..1] \rightarrow \{true, false\}$

$$discard : (\mathbf{p}, \iota) \mapsto visTex(\mathbf{p}) < \iota. \quad (26)$$

Note that even though the visibility function may be continuous, the thresholding operation gives a binary result and therefore no semi-transparent pixels appear, which avoids blending and the costly ordering of fragments.

By using different visibility textures, one can control in which way the individual fragments of a given object become visible. Examples include a uniform noise pattern, a function that decreases from the center outward, or any other function best suited to a given object. This has the effect that the amount and distribution of the visible fragments of an object can be controlled (see Figure 29). Also note that although *visTex* is given as a 3D function it is often not necessary to store it in a 3D texture, as can be seen by the noise texture example.

© The Eurographics Association 2011.

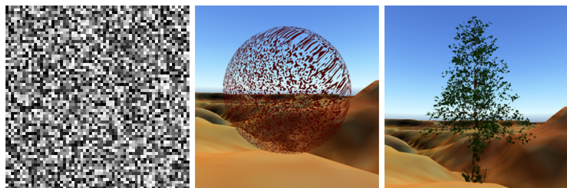

Figure 29: *A uniform noise visibility texture (*left*) applied to two different models with visibility* $\iota = 0.5$*).*

5.10 Streaming

Remote rendering

Remote rendering is a current trend that receives an increasing attention. Many companies such as OnLive, OTOY or Gaikai focus on the particular topic of game streaming. In such a scenario it is more than important to exploit TC in order to reduce bandwidth and computational effort on the server side.

In most cases the underlying technology for game streaming is closely related to video compression, with a few exceptions that transfer API calls directly to the client [NDS*08], but such an architecture assumes very advanced client hardware that can deal with all rendering commands.

Video-encoded rendering does not require a powerful client, but the bandwidth requirements can be high. Hence, temporal redundancy and perceptual limitations are a crucial component for such encoding algorithms. E.g., it is possible to exploit the reduced accuracy of the human visual system to pre-filter in-between frames to reduce the required bandwidth [FB08].

Usually, video encoding makes use of so-called I-frames that are only internally encoded and produce precise movie information. These I-frames are rare and completed by P-frames that rely on previous, and B-frames that make use of previous and following images. The latter type delivers quality-wise superior results, but is difficult to exploit for real-time applications. Because of the dependency on future frames, a delay is enforced which can be particularly problematic for lower frame rates. Furthermore, in the extreme case, if a frame drop occurs, very noticeable artifacts can arise.

A complete survey of video encoding goes clearly beyond the scope of this document. Here, we will describe some particular insights that relate to 3D rendering. Video compression for rendering should exploit the particularity of the content. One example is that many attributes can be extracted from the 3D scene itself which can then be used to improve the compression algorithms. One example is to accelerate the encoding process through the use of object-motion vectors [FE09] that are applied to predict pixel motion. The TC of the animation in the scene is, hereby, directly exploited. This solution can be very successful and enable higher compression [WKC94] than standard matching techniques.

One can even go further and rely on the scene attributes for reconstruction purposes. In fact, the previously discussed spatio-temporal upsampling strategies (Section 5.6) are very good candidates for application in a streaming context [PHE*11]. Such a combination has several advantages. Not only the bandwidth, also the server workload is tackled (only small resolution images are produced and transferred). Furthermore, as the previous frame is still present on the client when the new frame is supposed to be reconstructed, the algorithm can exploit this knowledge during compression and rely on these values as predictors for statistics-based encoding schemes [PHE*11].

This field of research is still relatively novel and it is likely to evolve significantly, but the mentioned recent advances illustrate the importance of exploiting TC in this context.

Large-data visualization

One particularly challenging field are large data sets. Here, content streaming is a topic that becomes increasingly important, especially due to the wealth of scanned data that often is tremendous in size. These difficult-to-render data sets exceed available memory capacities by far.

Usually, data structures are used to decompose the original data set in a hierarchical manner. The idea is to use structures that can be selectively refined. Once the right refinement scale is established for a view, only local modifications are applied that update the structure for the next frame. This lazy update scheme implicitly exploits TC because the modifications from one frame to the other can often be drastically limited. In some cases, even movement prediction can prove successful [LKR*96]. In any case, deriving an entirely new refinement would lead to a huge performance overhead.

These data structure hierarchies are further designed in a flexible way, in the sense that the actual geometric information is only transferred into them, when there is a request for it during rendering. In fact, not all data is needed at each point in time, as for a given viewpoint, occlusion can be exploited and unnecessary details omitted, which leads to a much smaller data subset that still produces a complete image.

In a STAR, this topic cannot be completely explored in depth and solutions exist for many different types of input data, varying from geometric models [WDS04], over point clouds [WBB*07], to recent volume-rendering approaches [GMAG08, CNLE09].

To illustrate the principle, we will base our discussion on ray-tracing queries. The main observation is that ray tracing is a useful tool, not only to produce images, but also to determine data fetches [WDS04]. Typically, scenes are organized in form of a tree (Figure 30 illustrates several levels of detail corresponding to levels in this tree). Rays then traverse the

© The Eurographics Association 2011.

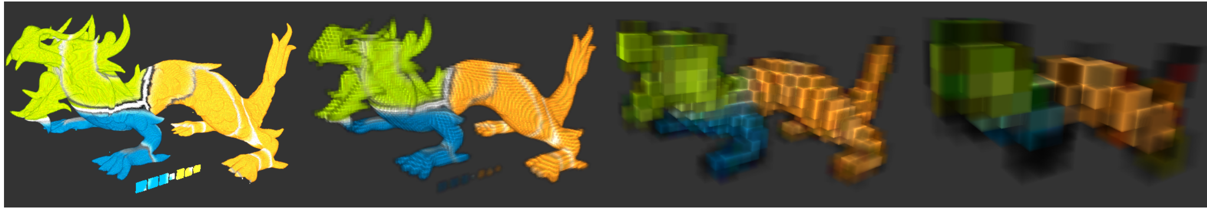

Figure 30: *At the basis of volume-data streaming algorithms is a hierarchical scene representation. Here, several levels of detail are illustrated. The resolution is switched automatically during the ray-tracing step, depending on the distance. Not all data is in memory at once, only the parts that are actually currently under use reside in memory.*

tree and test geometry intersections in each traversed node. The idea is that initially each node of the tree can be empty and will only be filled progressively during rendering. Whenever a ray reaches such an empty node, a data request is triggered and the ray potentially stopped, or traced against a simplified representation that fits into memory. In this way, the rays themselves control the level of detail, as well as frustum culling, or occlusion tests. No special handling of acceleration techniques is needed and, in particular, as rays tend to vary little from one view to the next (e.g., for a pure rotation of the camera many rays remain almost unchanged), the temporal redundancy is implicitly handled.

Such strategies have proven particularly efficient in the context of volume rendering [GMAG08, CNLE09]. Here, a multi-resolution data representation is arranged in the tree and whenever data is missing, rays do not need to be canceled, but can instead walk up the tree to access lower-resolution versions of the data. To deal with the memory constraints, such algorithms typically employ an LRU-cache mechanism (least-recently used), i.e., newly loaded elements will replace those that have not been accessed for a longer time. As elements tend to be active over coherent periods of time, such strategies prove particularly useful and can be implemented very efficiently on modern GPUs [CNSE10].

5.11 Online occlusion culling

Culling techniques like view-frustum culling [AM00] and visibility culling are important acceleration techniques for rasterization-based real-time rendering. View-frustum culling simply prunes all objects in the scene which don't intersect the current view frustum. Visibility culling prunes all objects that are occluded by other objects as early as possible in the pipeline. Using visibility culling we can achieve *output-sensitivity*, in the sense that the render time depends only on the complexity of the actually visible objects, not the complexity of the whole scene.

Visibility can be preprocessed by subdividing the view space into volumetric regions known as *view cells* and computing the potentially visible sets (PVSs) for each view cell in a lengthy offline step. An attractive alternative is online occlusion culling, where visibility is computed on the fly for the current view point. Online occlusion culling does not require a preprocess to store visibility information, and naturally allows for dynamic scenes. The major challenge for any online culling algorithm is to reduce the overhead caused by the online visibility calculations (the *occlusion queries*). A common use of occlusion queries is to conservatively test the visibility of a simple *proxy geometry*, e.g., the bounding box of a more complex object.

The importance of TC for online occlusion culling cannot be stressed enough - in fact utilizing TC is one of the key concepts to make online occlusion culling feasible in practice (together with using spatial hierarchies to exploit spatial coherence). Naively querying all objects leads to many wasted queries. The query overhead can become unacceptable in situations when most objects in the scene are visible. Therefore we exploit TC, assuming that objects that are (in)visible in one frame are likely to remain (in)visible in future frames. Using TC, we can substantially reduce the number of issued occlusion queries, as well as hide their latency.

5.11.1 Exploiting TC

The following general strategy is implemented in different forms by all state-of-the-art occlusion culling algorithms: First we establish a *visible front* by rendering those objects that were visible in the previous frame. Then we query the visibility of the previously invisible objects against this visible front. At last, to keep the overdraw low, we have to update the visibility classifications of the objects from the visible front. This can be done in a lazy manner, e.g., by querying each object every n frames (assuming coherence over several frames).

The clever hierarchical z-buffer algorithm proposed by [GKM93] uses both spatial hierarchies and TC in the manner described above for maximal efficiency. To accelerate visibility queries, it maintains a two-fold hierarchy - an image pyramid over the z-buffer and an octree hierarchy over the objects. The feasibility of this algorithm suffers from the drawback that only parts of it are supported by the hardware. The conceptually related algorithm of [ZMHI97] aims to speed up the queries by utilizing fast texture hardware.

© The Eurographics Association 2011.

5.11.2 Hardware occlusion queries

Since the GeForce3 hardware accelerated occlusion queries are supported on consumer graphics hardware. Hardware occlusion queries can be issued for a batch of rendered geometry. The query result is fetched with another command and simply returns the number of visible pixels. While hardware occlusion queries are fast, the queries still come with a non-negligible cost, and they have a certain latency until the query result is available on the CPU. Algorithms have to find a way to fill this latency in a meaningful way. A naive hierarchical implementation which waits for the query result at each node before further traversing a hierarchy can actually *slow down* rendering significantly due to CPU stalls – this is where TC comes into play.

5.11.3 Coherent hierarchical culling (CHC)

The *coherent hierarchical culling* (CHC) algorithm [BWPP04] utilizes TC to avoid such CPU stalls. This algorithm is conceptually simple and intuitive, while providing good performance in standard cases of moderate occlusion. It works with any kind of hierarchy that stores the geometry in the leaves. To establish the visible front, the CHC algorithm traverses hierarchy nodes in a front-to-back order.

The algorithm exploits temporal and spatial coherence by identifying invisible subtrees. To avoid wasted interior node queries, it starts issuing queries at the previous cut in the hierarchy (i.e., it queries previously invisible subtrees and visible leaves).

Furthermore, CHC assumes that previously visible leaves stay visible, and *never waits* for their query result. Instead, these nodes are *always* rendered in the current frame. Their visibility classifications are updated for the *next frame* once the result is available. For this purpose the pending queries are managed in a dedicated *query queue*. Fortunately, hardware occlusion queries provide a cheap way to check if a query result is available. This way, we can do some traversal and rendering on the CPU while the GPU is busy computing the query results, avoiding CPU stalls and GPU starvation.

5.11.4 Making further use of temporal coherence

The original CHC algorithm works sufficiently well in many situations, but still suffers from considerably overhead because of a) the large overall number of queries, b) the relatively high cost of individual queries.

The CHC++ algorithm [MBW08] addresses the before mentioned drawbacks by making better use of temporal and spatial coherence. It extends the CHC algorithm with a couple of simple but effective optimizations, leading to a speedup of 2-3 times compared to the previous state-of-the-art. The most important optimizations are:

Queues for batching of queries: A huge portion of the query cost in CHC is caused by the GPU state changes due to the constant interleaving of render and query mode (e.g., depth write on / off). Hence we aim to issue *batches of queries* instead of individual queries. For this purpose we append a node to a so called query candidate queue before it is queried, and then issue many queries at once. This reduces the number of state changes by one to two orders of magnitude. Note that we exploit coherence among (previously invisible) nodes in a batch, as we assume that mutual occlusion of these nodes is not relevant.

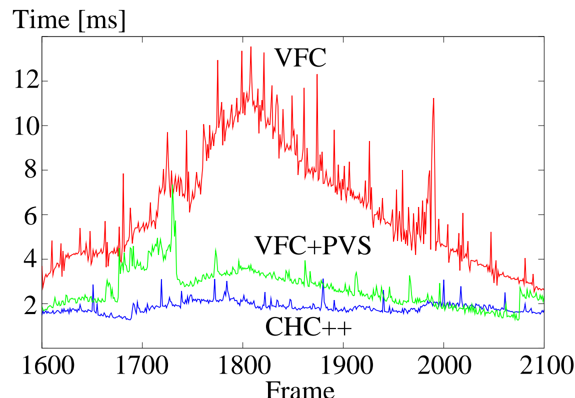

Figure 31: *Comparison of view frustum culling (VFC), view frustum culling and potentially visible sets (VFC+PVS), and CHC++ [BMW*09].*

Multiqueries: CHC++ compiles *multiqueries*, which are able to cover more nodes by a single occlusion query. This method is able to reduce the number of queries for previously invisible nodes up to an order of magnitude by making better use of TC. The decision of including a node in a multiquery is based on its history. I.e., nodes that were invisible for a long time are likely to stay invisible, hence they can be handled by a single query. Note that the nodes can be spatially completely unrelated.

Randomized sampling pattern for visible nodes: The algorithm applies a temporally jittered sampling pattern for scheduling queries for previously visible nodes. This method reduces the number of queries for visible nodes and while spreading them evenly over the frames of the walkthrough. It avoids potential frame rate drops that happen because of many queries being issued in the same frame due to simultaneous visibility changes (e.g., all rooftops of a town become visible in the same frame).

Figure 31 shows timings in the Powerplant scene with 12M triangles (using a NVIDIA GeForce 280 GTX). Interestingly, online occlusion culling with CHC++ is faster than rendering based on potentially visible sets (PVSs) in this walkthrough. In moderately to highly occluded scenarios, the overhead of the online occlusion culling algorithm typically is less than the overhead caused by the greater conservativity of

© The Eurographics Association 2011.

preprocessed visibility (as view cells under a certain size are not feasible).

5.12 Temporal perception

This report presented several algorithms that exploit TC of data. In other words, the redundancy of information over time. But in fact, TC can also be used in an inverse manner to produce richer content by exploiting elements of perception which seems to be a very promising avenue for future endeavors. Here, we will focus on two examples (color and resolution increase) that represent first steps in this direction of research.

One of the oldest examples to enrich graphics by exploiting temporal effects is to rely on flickering to increase the computer's color palette. The most prominent representatives of such a technique are DLP projectors that display, in a coherent way, the three color channels of an image in rapid succession. These separate signals are then integrated by the eye so that an observer perceives a fully colored image.

Similar to the DLP principle one can increase the available colors of a screen or machine. Back when color palettes were limited, having darkened tints of colors (e.g., for shadows) was not always possible. By flickering the corresponding elements on the screen a simple solution to extend the palette was born. For an observer these flickered colors mix because at higher frame rates the eye no longer distinguishes each frame individually. The same procedure is often employed in LCD screens under the name of *Frame Rate Control*. In practice, the material is often limited to 6 bits per color channel, whereas the graphics card produces 8-bit color channels. The solution is to represent fractional colors by displaying the immediate neighbors in quick succession over time [Art04]. Again, the eye integration delivers the illusion that what was observed on the retina is actually the fractional color value.

Besides more colors, also resolution and details can be addressed. It is known that object discrimination is more successful for subpixel camera panning than for corresponding static frames [KDT05, BSH06]. Didyk et al. [DER*10a] pushed this observation further by exploiting the temporal coherence of eye movement for apparent resolution enhancement. In other words, they are able to produce the illusion of high resolution on a low resolution screen and, hereby, even surpass the physical boundaries. Precisely, their setup is a low-resolution screen on which moving content is displayed at a high-refresh rate. When the eye starts tracking the information on the screen, several frames will be successively integrated on the retina. By predicting the eye movement it is possible to derive an image sequence such that the integrated response on the retina approaches a high-resolution image content. The accuracy of the tracking is assured by the human visual system's smooth-pursuit eye motion. This mechanism leads to an almost perfect stabilization for steady linear motion with velocities in the range of $0.625 - 2.5^{\circ}/s$ [LRP*06]. The low-resolution image sequence itself is derived using an optimization framework that takes eye integration and flicker perception into account to ensure that this sequence integrates properly on the retina.

6 Conclusion

In this report, we have described real-time rendering techniques that take advantage of temporal coherence by reusing expensive calculations from previously rendered frames. As a result, both performance and quality of many common real-time rendering tasks can be improved.

We started by showing that real-time rendering applications exhibit a significant amount of spatio-temporal coherence, thus motivating data reuse in shading computations. We then briefly surveyed the historical approaches focused on off-line methods before describing the real-time techniques which constitute the main focus of this report. We introduced the basic algorithm for performing real-time reprojection on the GPU. The approach allows the shader to efficiently query shading results from an earlier rendered frame (reverse reprojection), or similarly, map a shading result from the current frame to the next frame (forward reprojection). We then analyzed the quality vs. speed tradeoffs associated with data reuse.

We presented several applications that take advantage of data reuse. We started with the basic application of directly reusing results of an expensive shading computation, such as a procedural noise shader, from an earlier frame time. For applications that accumulate results from multiple renderings of the same scene, such as stereo, motion blur, and depth-of-field rendering, we showed how to reuse shading results from a "central frame" when rendering the remaining accumulated frames, thereby reducing rendering times considerably.

Some expensive per-pixel computations often require approximating an integral by combining multiple spatial samples, such as shadow computation. To address those scenarios, we described how to amortize computation by combining results from multiple frames in order to achieve better results for antialiasing, pixel correct shadows, and soft shadows, among other applications. Using reprojection for these techniques allows for a much larger number of samples for the same rendering time budget. The amortized approach also allows for a smoothly varying shading result instead of the "all or nothing" reuse strategy of the earlier applications that either fully reuse the earlier results or compute it entirely anew. We showed significant improvement in quality and speed for these amortized approaches and analized the tradeoff between lag and aliasing in the rendered result. We also showed how TC can be used to compute not only efficient shadows from a light source, but also efficient global illumination approximations through amortization. We then presented techniques to combine both spatial and temporal upsampling using a joint bilateral filter that considers samples from recent rendered frames, and how to increase framerates by generating new intermediate frames by taking advantage of TC.

Finally, we showed how TC can be used to improve quality

© The Eurographics Association 2011.

or accelerate a variety of tasks. Forward reprojection was applied to smoothly advect brush strokes for non-photorealistic rendering of animated scenes. TC was used to render transition phases for discrete LOD blending, thereby avoiding popping artifacts and creating a smooth transition between levels of detail. We then showed how TC has also been explored for streaming content, such as improved compression for remote rendering of synthetic scenes (e.g., games), and large-data visualization. Another area that has been explored is how to use TC to accelerate occlusion culling. Finally, we showed techniques that consider elements of perception of the human visual system in order to increase the apparent number of colors and apparent resolution of the image.

To summarize, this report surveyed strategies for reusing shading computations during real-time rendering. These strategies are very generally applicable as demonstrated on a very large number of different application scenarios. While relatively recent, this research trend has already found uses in the gaming community. With the continued increase in complex shading effects, frame rates, screen resolution, and rendering hardware features, we expect that techniques that take advantage of temporal coherence will become even more prevalent.

References

[AH93] Adelson S. J., Hodges L. F.: Stereoscopic Ray-Tracing. *The Visual Computer 10*, 3 (1993), 127–144. 10

[AH95] Adelson S. J., Hodges L. F.: Generating exact ray-traced animation frames by reprojection. *IEEE Comput. Graph. Appl. 15*, 3 (1995), 43–52. 3

[AM00] Assarsson U., Möller T.: Optimized view frustum culling algorithms for bounding boxes. *Journal of graphics, GPU, and game tools 5*, 1 (2000), 9–22. 21

[And10] Andreev D.: Real-time frame rate up-conversion for video games. In *ACM SIGGRAPH 2010 Talks* (7 2010). 5, 18

[Art04] Artamonov O.: X-bit's guide: Contemporary lcd monitor parameters and characteristics. http://www.xbitlabs.com/articles/monitors/display/lcd-guide_11.html, October 2004. 23

[ATS94] Arvo J., Torrance K., Smits B.: A framework for the analysis of error in global illumination algorithms. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 75–84. 14

[BFMZ94] Bishop G., Fuchs H., McMillan L., Zagier E. J. S.: Frameless rendering: double buffering considered harmful. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 175–176. 3

[BJ88] Badt Jr. S.: Two algorithms for taking advantage of temporal coherence in ray tracing. *VC 4* (1988), 123–132. 2

[BM05] Bennett E. P., McMillan L.: Video enhancement using per-pixel virtual exposures. *ACM Transactions on Graphics 24*, 3 (2005), 845–852. 17

[BMW*09] Bittner J., Mattausch O., Wonka P., Havran V., Wimmer M.: Adaptive global visibility sampling. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Papers* (New York, NY, USA, 2009), ACM. 22

[BSD08] Bavoil L., Sainz M., Dimitrov R.: Image-space horizon-based ambient occlusion. In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks* (2008). 14

[BSH06] Bijl P., Schutte K., Hogervorst M. A.: Applicability of TOD, MTDP, MRT and DMRT for dynamic image enhancement techniques. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* (2006), vol. 6207. 23

[BWPP04] Bittner J., Wimmer M., Piringer H., Purgathofer W.: Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum 23*, 3 (Sept. 2004), 615–624. Proceedings EUROGRAPHICS 2004. 22

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)* (Boston, MA, Etats-Unis, feb 2009), ACM, ACM Press. to appear. 20, 21

[CNSE10] Crassin C., Neyret F., Sainz M., Eisemann E.: *Efficient Rendering of Highly Detailed Volumetric Scenes with GigaVoxels. In book: GPU Pro*. A K Peters, 2010, ch. X.3, pp. 643–676. 21

[CT81] Cook R. L., Torrance K. E.: A reflectance model for computer graphics. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1981), ACM, pp. 307–316. 14

[CW93] Chen S. E., Williams L.: View interpolation for image synthesis. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), ACM, pp. 279–288. 10

[DDM03] Damez C., Dmitriev K., Myszkowski K.: State of the art in global illumination for interactive applications and high-quality animations. *Computer Graphics Forum 22*, 1 (Mar. 2003), 55–77. 14

[DER*10a] Didyk P., Eisemann E., Ritschel T., Myszkowski K., Seidel H.-P.: Apparent display resolution enhancement for moving images. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2010, Los Angeles) 29*, 3 (2010). 23

[DER*10b] Didyk P., Eisemann E., Ritschel T., Myszkowski K., Seidel H.-P.: Perceptually-motivated real-time temporal upsampling of 3D content for high-refresh-rate displays. *Computer Graphics Forum 29*, 2 (2010), 713–722. 1, 5, 18

[DRE*10] Didyk P., Ritschel T., Eisemann E., Myszkowski K., Seidel H.-P.: Adaptive image-space stereo view synthesis. In *Proc. Vision, Modeling and Visualization Workshop* (11 2010). 5, 10

[ED04] Eisemann E., Durand F.: Flash photography enhancement via intrinsic relighting. In *ACM Transactions on Graphics (Proceedings of Siggraph Conference)* (2004), vol. 23, ACM Press. 17

[ED07a] Eisemann E., Décoret X.: On exact error bounds for view-dependent simplification. *Comput. Graph. Forum 26*, 2 (2007), 202–213. 3

[ED07b] Eisemann E., Décoret X.: Visibility sampling on gpu and applications. *Computer Graphics Forum (Proceedings of Eurographics 2007) 26*, 3 (2007). 13

[FB08] Fujibayashi A., Boon C. S.: A masking model for motion sharpening phenomenon in video sequences. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A*, 6 (2008), 1408–1415. 20

[FC08] Fox M., Compton S.: Ambient occlusive crease shading. *Game Developer Magazine (March 2008)* (March 2008). 14

[FE09] Fechteler P., Eisert P.: Depth map enhanced macroblock partitioning for H.264 video coding of computer graphics content. In *Intern. Conf. on Image Proc.* (2009), pp. 3441–3444. 20

[FPD08] Feng X.-F., Pan H., Daly S.: Comparisons of motion-blur assessment strategies for newly emergent LCD and backlight driving technologies. *Journal of the Society for Information Display 16* (2008), 981–988. 18

[GKM93] Greene N., Kass M., Miller G.: Hierarchical Z-buffer visibility. In *Computer Graphics (Proceedings of SIGGRAPH '93)* (1993), pp. 231–238. 21

[GMAG08] Gobbetti E., Marton F., Antonio J., Guitian I.: A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *Vis. Comput. 24*, 7 (2008), 797–806. 20, 21

[GW06] Giegl M., Wimmer M.: Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. *Computer Graphics Forum 26*, 1 (Mar. 2006), 46–49. 19

[HA90] Haeberli P., Akeley K.: The accumulation buffer: hardware support for high-quality rendering. In *Proc. SIGGRAPH '90* (1990), ACM, pp. 309–318. 10

[HBS03] Havran V., Bittner J., Seidel H.-P.: Exploiting temporal coherence in ray casted walkthroughs. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics* (New York, NY, USA, 2003), ACM Press, pp. 149–155. 3

[HDMS03] Havran V., Damez C., Myszkowski K., Seidel H.-P.: An efficient spatio-temporal architecture for animation rendering. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (2003), Springer, pp. 106–117. 10

[HEMS10] Herzog R., Eisemann E., Myszkowski K., Seidel H.-P.: Spatio-temporal upsampling on the GPU. In *Symposium on Interactive 3D Graphics and Games* (2010), ACM. 17

[HH97] Heckbert P. S., Herf M.: *Simulating Soft Shadows with Graphics Hardware*. Tech. Rep. CMU-CS-97-104, CS Dept., Carnegie Mellon U., Jan. 1997. CMU-CS-97-104, http://www.cs.cmu.edu/ ph. 12

[Jan01] Janssen R.: *Computational Image Quality*. Spie Press, Bellingham, Washington USA, 2001. 18

[Kaj86] Kajiya J. T.: The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 143–150. 14

[KCLU07] Kopf J., Cohen M. F., Lischinski D., Uyttendaele M.: Joint bilateral upsampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) 26*, 3 (2007). 17

[KDT05] Krapels K., Driggers R. G., Teaney B.: Target-acquisition performance in undersampled infrared imagers: static imagery to motion video. *Applied Optics 44*, 33 (2005), 7055–7061. 23

[Kel97] Keller A.: Instant radiosity. In *Proceedings of SIGGRAPH 97* (Aug. 1997), Computer Graphics Proceedings, Annual Conference Series, pp. 49–56. 16

[KTM*10] Knecht M., Traxler C., Mattausch O., Purgathofer W., Wimmer M.: Differential instant radiosity for mixed reality. In *Proc. Ninth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'10)* (Seoul, Korea, October 2010). 16

[KV04] Klompenhouwer M. A., Velthoven L. J.: Motion blur reduction for liquid crystal displays: Motion-compensated inverse filtering. In *Proc. SPIE, Vol. 5308, 690* (2004). 17

[Lan02] Landis H.: Production-ready global illumination. In *Proceedings of the conference on SIGGRAPH 2002 course notes 16* (2002). 14

[LKR*96] Lindstrom P., Koller D., Ribarsky W., Hodges L. F., Faust N., Turner G. A.: Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH* (1996), pp. 109–118. 20

[LRP*06] Laird J., Rosen M., Pelz J., Montag E., Daly S.: Spatio-velocity CSF as a function of retinal velocity using unstabilized stimuli. In *Human Vision and Electronic Imaging XI* (2006), vol. 6057 of *SPIE Proceedings Series*, pp. 32–43. 23

[LS97] Lengyel J., Snyder J.: Rendering with coherent layers. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 233–242. 3

[LSF10] Lu J., Sander P. V., Finkelstein A.: Interactive painterly stylization of images, videos and 3d animations. In *Proc. Symposium on Interactive 3D Graphics and Games* (2010), pp. 127–134. 18

[LSK*07] Laine S., Saransaari H., Kontkanen J., Lehtinen J., Aila T.: Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007* (2007), Eurographics Association, pp. 277–286. 16

[MB95] McMillan L., Bishop G.: Head-tracked stereoscopic display using image warping. In *Proceedings SPIE, volume 2409* (1995), pp. 21–30. 10

[MBW08] Mattausch O., Bittner J., Wimmer M.: Chc++: Coherent hierarchical culling revisited. *Computer Graphics Forum (Proceedings of Eurographics 2008) 27*, 3 (Apr. 2008), 221–230. 22

[Mit07] Mittring M.: Finding next gen - cryengine 2. In *Proceedings of the conference on SIGGRAPH 2007 course notes, course 28, Advanced Real-Time Rendering in 3D Graphics and Games* (2007), ACM Press, pp. 97–121. 14

[MKC07] Marroquim R., Kraus M., Cavalcanti P. R.: Efficient point-based rendering using image reconstruction. In *Proc. Eurographics Symposium on Point-Based Graphics* (2007), pp. 101–108. 5

[MSW10] Mattausch O., Scherzer D., Wimmer M.: High-quality screen-space ambient occlusion using temporal coherence. *Computer Graphics Forum 29(8)* (2010), 2492–2503. 14

[NDS*08] Nave I., David H., Shani A., Laikari A., Eisert P., Fechteler P.: Games@Large graphics streaming architecture. In *Symp. on Consumer Electronics (ISCE)* (2008). 20

[NSL*07] Nehab D., Sander P. V., Lawrence J., Tatarchuk N., Isidoro J. R.: Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware* (2007), pp. 25–35. 4, 5, 6, 7, 9, 10

[PFD05] Pan H., Feng X.-F., Daly S.: LCD motion blur modeling and analysis. In *Proc. ICIP* (2005), pp. 21–24. 17

[PHE*11] Pajak D., Herzog R., Eisemann E., Myszkowski K., Seidel H.-P.: Scalable remote rendering with depth and motion-flow augmented streaming. *Computer Graphics Forum 30*, 2 (2011). Proc. of Eurographics. 20

[PSA*04] Petschnigg G., Szeliski R., Agrawala M., Cohen M., Hoppe H., Toyama K.: Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics (Proceedings of Siggraph Conference) 23*, 3 (2004), 664–672. 17

[QWQK00] Qu H., Wan M., Qin J., Kaufman A.: Image based rendering with stable frame rates. In *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)* (Washington, DC, USA, 2000), IEEE Computer Society. 3

© The Eurographics Association 2011.

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA 2008) 27*, 5 (2008), 129. 16

[RP94] REGAN M., POSE R.: Priority rendering with a virtual reality address recalculation pipeline. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 155–162. 3

[SaLY*08a] SITTHI-AMORN P., LAWRENCE J., YANG L., SANDER P. V., NEHAB D.: An improved shading cache for modern GPUs. In *Proc. of Graphics Hardware* (6 2008), pp. 95–101. 6, 9

[SaLY*08b] SITTHI-AMORN P., LAWRENCE J., YANG L., SANDER P. V., NEHAB D., XI J.: Automated reprojection-based pixel shader optimization. *ACM Trans. Graph. 27*, 5 (12 2008), 127. 7, 9

[SB95] SMITH S. M., BRADY J. M.: *SUSAN – A new approach to low level image processing*. Tech. Rep. TR95SMS1c, Chertsey, Surrey, UK, 1995. 17

[Sch96] SCHAUFLER G.: Exploiting frame to frame coherence in a virtual reality system. In *VRAIS '96: Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS 96)* (Washington, DC, USA, 1996), IEEE Computer Society, p. 95. 3

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample-based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2008) 27*, 4 (June 2008), 1285–1292. 13

[SGHS98] SHADE J., GORTLER S., HE L.-W., SZELISKI R.: Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), ACM, pp. 231–242. 3

[Shi95] SHINYA M.: Improvements on the Pixel-tracing Filter: Reflection/Refraction, Shadows, and Jittering. In *Graphics Interface '95* (1995), pp. 92–102. 17

[SJW07] SCHERZER D., JESCHKE S., WIMMER M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Eurographics Symposium on Rendering* (2007), pp. 45–50. 4, 6, 12

[SKUT*10] SZIRMAY-KALOS L., UMENHOFFER T., TOTH B., SZECSI L., SBERT M.: Volumetric ambient occlusion for real-time rendering and games. *IEEE Computer Graphics and Applications 30* (2010), 70–79. 14

[SLS*96] SHADE J., LISCHINSKI D., SALESIN D. H., DEROSE T., SNYDER J.: Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 75–82. 3

[SS00] SIMMONS M., SEQUIN C. H.: Tapestry: A dynamic mesh-based display representation for interactive rendering. In *Proceedings of the 11th Eurographics Workshop on Rendering* (2000), pp. 329–340. 3

[SSMW09] SCHERZER D., SCHWÄRZLER M., MATTAUSCH O., WIMMER M.: Real-time soft shadows using temporal coherence. *Lecture Notes in Computer Science (LNCS)* (Nov. 2009). 13

[SSS74] SUTHERLAND I. E., SPROULL R. F., SCHUMACKER R. A.: A characterization of ten hidden-surface algorithms. *ACM Comput. Surv. 6*, 1 (1974), 1–55. 2

[SW08] SCHERZER D., WIMMER M.: Frame sequential interpolation for discrete level-of-detail rendering. *Computer Graphics Forum (Proceedings EGSR 2008) 27*, 4 (June 2008), 1175–1181. 19

[SW09] SMEDBERG N., WRIGHT D.: Rendering techniques in gears of war 2, 2009. 14

[Tek95] TEKALP A. M.: *Digital video Processing*. Prentice Hall, 1995. 17

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *ICCV* (1998), pp. 839–846. 17

[TV05] TAKEUCHI T., VALOIS K. D.: Sharpening image motion based on the spatio-temporal characteristics of human vision. In *Proc. SPIE, Vol. 5666, 690* (2005), pp. 83–94. 18

[VALBW06] VELÁZQUEZ-ARMENDÁRIZ E., LEE E., BALA K., WALTER B.: Implementing the render cache and the edge-and-point image on graphics hardware. In *GI '06: Proceedings of Graphics Interface 2006* (Toronto, Ont., Canada, Canada, 2006), Canadian Information Processing Society, pp. 211–217. 3

[WBB*07] WAND M., BERNER A., BOKELOH M., FLECK A., HOFFMANN M., JENKE P., MAIER B., STANEKER D., SCHILLING A.: Interactive editing of large point clouds. In *Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings* (Prague, Czech Republik, 2007), Chen B., Zwicker M., Botsch M., Pajarola R., (Eds.), Eurographics Association, pp. 37–46. 20

[WDG02] WALTER B., DRETTAKIS G., GREENBERG D. P.: Enhancing and optimizing the render cache. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 37–42. 3

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive rendering using the render cache. In *Rendering techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)* (New York, NY, Jun 1999), Lischinski D., Larson G., (Eds.), vol. 10, Springer-Verlag/Wien, pp. 235–246. 3

[WDS04] WALD I., DIETRICH A., SLUSALLEK P.: An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Proceedings of the Eurographics Symposium on Rendering* (2004). (to appear). 20

[WGS99] WIMMER M., GIEGL M., SCHMALSTIEG D.: Fast walkthroughs with image caches and ray casting. In *Virtual Environments '99. Proceedings of the 5th Eurographics Workshop on Virtual Environments* (June 1999), Gervautz M., Schmalstieg D., Hildebrand A., (Eds.), Eurographics, Springer-Verlag Wien, pp. 73–84. ISBN 3-211-83347-1. 3

[WKC94] WALLACH D. S., KUNAPALLI S., COHEN M. F.: Accelerated MPEG compression of dynamic polygonal scenes. In *Proceedings of SIGGRAPH* (1994), pp. 193–197. 20

[YNS*09] YANG L., NEHAB D., SANDER P. V., SITTHI-AMORN P., LAWRENCE J., HOPPE H.: Amortized supersampling. *ACM Trans. Graph. 28*, 5 (2009), 135. 8, 9, 10, 11

[YWY10] YU X., WANG R., YU J.: Real-time depth of field rendering via dynamic light field generation and filtering. *Computer Graphics Forum (Proc. of Pacific Graphics) 29*, 7 (2010). 4, 10

[ZMHI97] ZHANG H., MANOCHA D., HUDSON T., III K. E. H.: Visibility culling using hierarchical occlusion maps. In *SIGGRAPH* (1997), pp. 77–88. 21

[ZWL05] ZHU T., WANG R., LUEBKE D.: A gpu-accelerated render cache. *Pacific Graphics, (Short Paper Session)* (October 2005). 3

© The Eurographics Association 2011.