An Overview of Temporal Coherence Methods in Real-Time Rendering

Daniel Scherzer

***Abstract*— Most of the power of modern graphics cards is put into the acceleration of shading tasks because here lies the major bottleneck for most sophisticated real-time algorithms. By using *temporal coherence*, i.e. reusing shading information from a previous frame, this problem can be alleviated. This paper gives an overview of the concepts of temporal coherence in real-time rendering and should give the reader the working practical and theoretical knowledge to exploit temporal coherence in his own algorithms.**

I. INTRODUCTION

One of the driving forces of computer graphics is to render physically correct images with rich visual effects. This usually requires large scenes with highly detailed models, as well as computationally intensive shading work to be incorporated in a modern rendering system. On the other hand, real-time rendering has the conflicting goal of creating a sequence of such images fast enough to still allow for continuous animation and user interaction. Here a limit of at least 60 frames per second is considered as sufficiently smooth for the human observer, which means the time available for one frame is about 16 milliseconds. All calculations necessary to create a frame have to fit into this time budget. This not only includes all the rendering algorithms we are concerned with in computer graphics, but may also contain the domain specific code of an application, artificial intelligence, input processing and sound rendering.

Although computer graphics hardware has made staggering advances in terms of speed and programmability, there still exist a number of algorithms that are too expensive to be computed in this time budget. A few important examples include physically correct shadows, depth of field and motion blur effects, or even an ambient occlusion approximation to the exact global illumination solution. The situation becomes worse when these effects are combined with large and complex scenes, in which the hidden geometry often consumes a significant portion of render time but contributes nothing to the final images.

One way to circumvent this hard time limit is to capitalize on *Temporal Coherence* (TC) and avoid redundant computations over time. TC is hereby defined as the existence of a correlation in time of the output of a given algorithm. For example, in a scene rendered at high frame rates, there is usually very little difference in the shading over visible surfaces between two consecutive frames, and the majority of surfaces are mutually visible (see Figure 1). Therefore, computing everything from scratch in every frame is potentially wasteful. Exploiting the coherence between adjacent frames and reuse intermediate or final shading result can therefore reduce the average shading cost of generating a single frame.

Daniel Scherzer is an Assistant Professor at the Institute of Computer Graphics and Algorithms, Vienna University of Technology
scherzer@cg.tuwien.ac.at

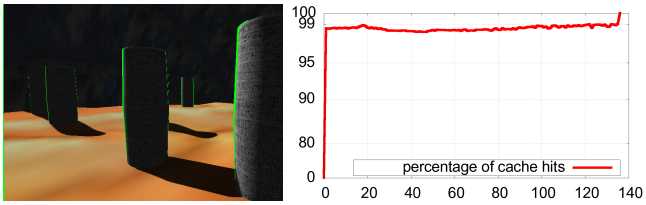

Fig. 1. Temporal coherence that exists in a game-like scene. *Left:* For a strafe-left movement the cache misses are shown in green. *Right:* Plot of the percentage of pixels found in the cache for each frame of the animation sequence.

In general, TC can be applied for achieving either of the following goals:

- *Acceleration*: A given algorithm can be accelerated by reformulating it as incremental in time, thereby amortizing the total workload over several frames. The output quality may be marginally degraded but the overall speed improvement is often promising.
- *Quality improvement*: The results of a given algorithm can be augmented by taking into account results computed in previous frames. By a slightly increase in render time, the quality of the result can often be significantly improved.
- *Reducing temporal aliasing*: When rendering frames, for each frame independent rasterizations are produced. This causes temporal aliasing and can result in strong flickering artifacts. TC can be applied to avoid this by introducing knowledge of previous rasterizations into the calculations for the current one, thereby allowing for temporal smoothing by dissallowing sudden changes in coherent regions.

These goals have in common that for a drastic change in the input some latency in the output may be introduced. In the acceleration case this requires a major refresh in the previously computed results, which may cause a sudden drop of framerate. In the quality improvement case, this means that over several frames only an approximate solution can be displayed before the algorithm converges. Fortunately with relatively high framerates and careful algorithmic designs, these problems can often be handled smoothly and unnoticed by the viewer. In addition, ongoing animation may also cause information from previous frames to be outdated. This has to be accounted for in order to avoid temporal artifacts such as after-images or tailing.

Aside from the fact that the redesign of algorithms to account for TC can be challenging, special care has to be taken to fit these algorithms to the massively parallel nature of modern graphics architectures.

II. BACKGROUND

The term *frame-to-frame coherence* was first introduced by Sutherland et al. [19] in his seminal paper "Characterization of Ten Hidden-Surface Algorithms", in which he describes various versions of coherence, like scan-line or area coherence that allow for more efficient rendering.

In *ray-tracing* reprojection of object space information can be used to allow to reuse information from the previous frame to accelerate animations [2], [1].

In *image-based rendering* TC allows to replace parts of a scene with image-based representations. Complex distant geometry can be replaced by so called impostors [6], [13]. A scene can also be factorized into multiple layers by accounting for differences in perception of fore-/background objects, as well as differences in the motion of objects [8]. The extreme case is frameless rendering, which only relies on TC instead of spatial coherence [3]. Each pixel is rendered independently based on the most recent input. Pixels stay visible for a random time-span. To avoid image tearing pixels are rendered in a random order.

In *image warping* images are used as a cache to be reused and warped into different views. In-between views can be calculated by morphing a number of reference images [4], [10], [9]. For each view color and depth are stored. Both images are warped into the new view to allow for small camera movements. Then the two images are composited together to compensate for most disocclusions [9]. A more involved representations is the *render cache* [22]. It is intended as an acceleration data structure for renderers that are too slow for interactive use. It is a point based structure, which stores previous results, namely 3d coordinates and shading information. By using reprojection, image space sparse sampling heuristics and by exploiting spatio-temporal image coherence these results can be reused in the current frame. Progressive refinement allows decoupling the rendering and display frame rates, enabling high interactivity. This approach was later extended with predictive sampling and interpolation filters [21] and finally accelerated on the GPU [20], [24].

III. IMAGE-SPACE REAL-TIME REVERSE REPROJECTION

Similar to Walters' render cache idea is Nehab et al.'s [11], [12] so-called *reprojection cache*, which is introduced as a way to accelerate real-time pixel shading in hardware rasterization renderers (see Figure 2). The main difference to the *render cache* is that the *reprojection cache* does not contain points but visible pixels in screen space (with additional data like depth). This is a very hardware friendly approach as this cache is just a viewport-sized off-screen buffer and can therefore reside in graphics memory without causing traffic between GPU and CPU. Another difference that fits perfectly to hardware is that this method uses reverse reprojection and

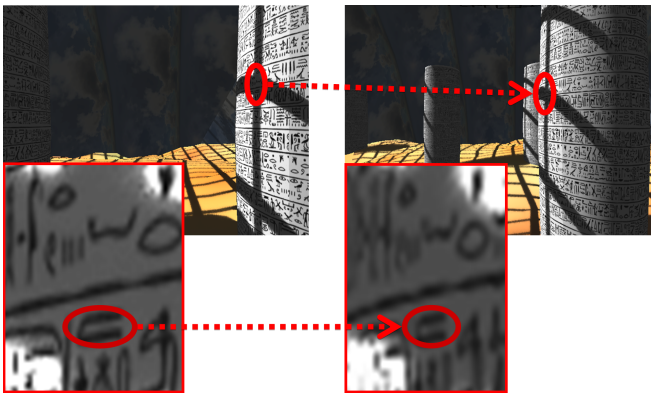

Fig. 2. Fast reprojection on the hardware is achieved by back-projecting each fragment from the current frame *(left)* into the cache *(right)* and incorporating the information found there to shade the current fragment.

therefore can use hardware texture filtering capabilities for sample retrieval. The *reprojection cache* approach described in this paper is similar to our concurrent work [14], where we specialized in improving shadow quality.

Reprojection is achieved by back-projecting each fragment $\mathbf{p}$ into the coordinate space of the previous frame – the space in which the reprojection cache was created. Consequently, if the camera moves, for every currently rendered fragment we have to find the corresponding position in the reproojection cache. Since we have the 3D position of our current fragment (in the post-perspective space of the current view), we can simply use the view ($\mathbf{V}$) and projection ($\mathbf{P}$) matrices and their inverses of the current and the last frame to do the transformation (back into the post-perspective space of the previous frame):

$$\mathbf{p}_{prev} = \mathbf{P}_{prev} * \mathbf{V}_{prev} * \mathbf{V}^{-1} * \mathbf{P}^{-1} * \mathbf{p} \qquad (1)$$

Here $\mathbf{p}$ is the fragment in the post-perspective space of the current frame. This fragment is transformed by $\mathbf{P}^{-1}$, the inverse projection matrix of the current frame, $\mathbf{V}^{-1}$, the inverse view matrix of the current frame (we are now in world space), $\mathbf{V}_{prev}$, the view matrix of the previous frame and finally by $\mathbf{P}_{prev}$, the projection matrix of the previous frame. After homogenization we are at the position the fragment would have had in the previous frame $\mathbf{p}_{prev}$. For moving objects, we can additionally store the object space transformation matrices or skinning matrices to do the backprojection step. Please note that all matrix transformations can be performed in the vertex shader and only the homogenization (division by the w-coordinate) has to happen in the fragment shader.

The obtained position will normally not be at an exact fragment center in the history buffer except for the special case that no movement has occurred. Consequently, filtering the history buffer for the lookup should be done. In practice, the bilinear filtering that graphics hardware offers shows good results.

For distinguishing between a cache hit and miss when reprojecting current fragments into the cache, the depth is used. If a cache value has a depth equal ($\pm\epsilon$) to the current

fragment's depth, a cache hit is assumed (see Figure 1) and information from the cache can be reused. Otherwise no temporally coherent information for this fragment is available.

Due to its speed and versatility this approach is the de facto standard for using TC in real-time rendering. In the following sections we will therefore discuss a selection of algorithms that are based on this method.

A. Discrete LOD Interpolation

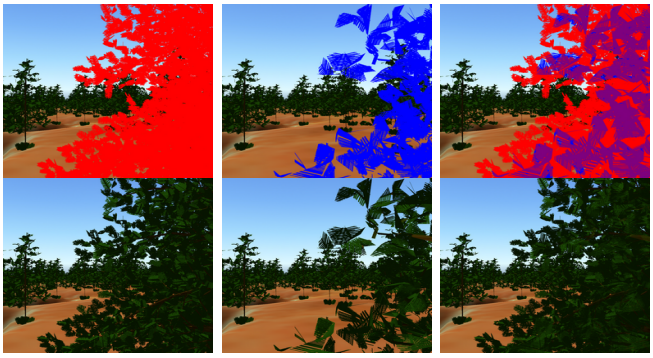

Fig. 3. LOD interpolation combines two buffers containing the discrete LODs to create smooth LOD transitions. *First and second column*: buffers; *last column*: combination. The top row shows the two LODs in red and blue respectively.

The idea of discrete level-of-detail (LOD) techniques is to use a set of representations with different levels of detail for one model and select the most appropriate representation for rendering at runtime. Due to memory constrains only a small number of LODs is being used and therefore switching from one representation to another can lead to noticeable popping artifacts. A solution to this problem is to replace the hard switch by a transition phase, in which the two representations are alpha blended in screen space [5].

Apart from other problems, this approach requires that the geometry (and the shaders) of both representations 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 Scherzer and Wimmer introduce LOD interpolation (see Figure 3) [16]. 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 we use one of the two LOD representations and render only a certain amount of its fragments (see Figure 4), depending one were in the transition (i.e., how visible) this object currently is. This is done via so-called *visibility textures*, which represent a visibility function for an object. In the next frame the same is done the other LOD representation and rendered 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.

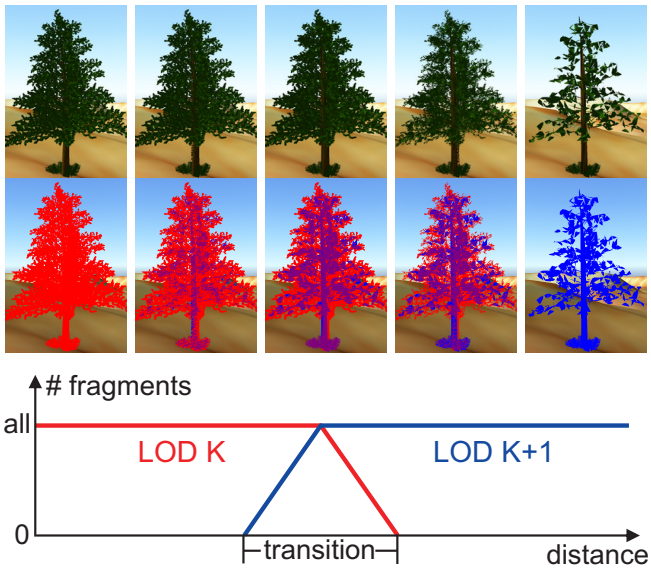

Fig. 4. Transition phase from LOD_k to LOD_{k+1}: *left:*LOD_k; *middle:* midway in the transition all fragments of both LODs are drawn; *right:* LOD_{k+1}; *Below:* First LOD_{k+1} is gradually introduced till all its fragments are drawn. Then LOD_k is gradually removed by rendering fewer and fewer fragments. The top two rows show the result of our method and a false color illustration.

We store a 3D visibility function per object (the *visibility texture*) and compare it to a visibility threshold to decide which fragments to discard. The visibility threshold $\tau \in [0..1]$ is given by the function depicted in Figure 4. Written as an equation: $\lambda : R^3 \times [0..1] \rightarrow \{true, false\}$

$$\lambda : (\mathbf{p}, \tau) \mapsto visTex(\mathbf{p}) > \tau \qquad (2)$$

is the function that evaluates for each fragment if it should be discarded. Here $\mathbf{p}$ is the object-space coordinate (before any animation is applied) of the fragment, τ the visibility threshold and $visTex$ is the lookup into the visibility texture. Note that even though the visibility function may be continuous, the thresholding operation gives a binary result and therefore no semi-transparent pixels appear.

By using different visibility textures, one can control in which way the fragments 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 5).

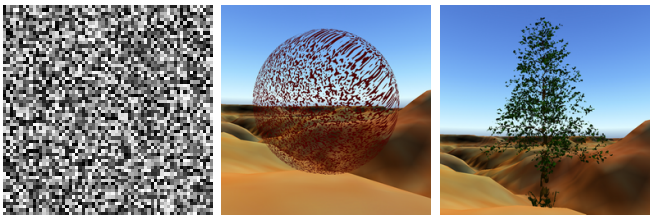

Fig. 5. A uniform noise visibility texture (*left*) applied to two different models with visibility $\tau = 0.5$).

B. Hard 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 6).

The most concerning visual artifacts of this method 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 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 6).

The main idea in [14] is to jitter the shadow map viewport differently in each frame and to combine the results over several frames, leading to a higher effective resolution.

In order to reduce temporal aliasing, each pixel is interpreted as a separate function $f(n)$ with the time as the input domain (usually represented by a frame number n). Temporal anti-aliasing is then done by smoothing this function. Smoothing itself is done by employing *exponential smoothing*:

$$s(n) = w * f(n) + (1 - w) * s(n - 1) \qquad 0 < w \leq 1 \quad (3)$$

Here w is a weight and $s(n-1)$ is the result of the previous evaluation. w allows balancing fast adaption of s to changing input parameters against temporal noise of the function. With increasing w, $s(n)$ depends more on the result of the current frame function and less on older frames and vice versa.

The shadow quality in this approach can actually be made to converge to a pixel-perfect result by optimizing the choice of the weight between the current and the previous frame result (stored in a reprojection cache). The weight is determined according to the *confidence* of the shadow lookup. The confidence is 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 7 and 8).

C. Soft Shadows

In reality most light sources are area light sources and hence most shadows exhibit soft borders. *Light source sampling* introduced by Heckbert and Herf [7] 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 for each fragment. The primary problem here is that the number of samples (and therefore shadow maps) to produce smooth penumbrae is huge and therefore this approach is slow. 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. 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

The main idea of our algorithm [15] is to formulate light source area sampling in an iterative manner, evaluating only a single shadow map per frame. We start by looking at the math for light source area sampling: Given n shadow maps, we can calculate the soft shadow result for a given pixel $\mathbf{p}$ by averaging over the hard shadow results s_i calculated for each shadow map. This is given by

$$\psi_n(\mathbf{p}) = \frac{1}{n} \sum_{i=1}^{n} s_i(\mathbf{p}). \quad (4)$$

We want to evaluate this formula iteratively by adding a new shadow map each frame, combining its shadow

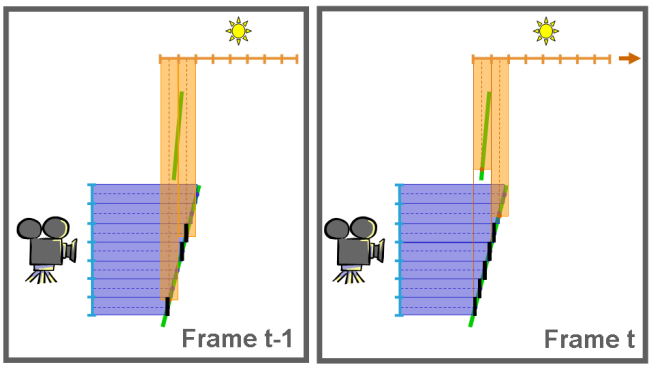

Fig. 6. If the rasterization of the shadow map changes (here represented by a right shift), the shadowing results may also change. On the *left* three fragments are in shadow, while on the *right* five fragments are in shadow. This results in flickering or swimming artifacts in animations.

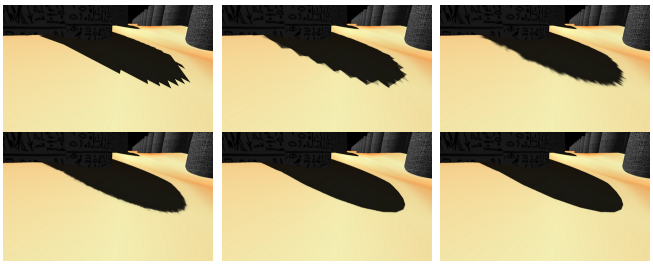

Fig. 7. Shadow adaption over time of an undersampled uniform shadow map after 0 *(top-left)*, 1 *(top-middle)*, 10 *(top-right)*, 20 *(bottom-left)*, 30 *(bottom-middle)* and 60 *(bottom-right)* frames.

information with the data from previous frames stored in a so-called shadow buffer B_{prev}, and storing it in a new shadow buffer B_{cur}. With this approach, the approximated shadow in the buffer improves from frame to frame and converges to the true soft shadow result (see Figure 9).

Our approach has the following steps:

- The area sampling is done one sample per frame by creating a shadow map from a randomly selected position on the area light. For each screen pixel the hard shadow results obtained from this shadow map are combined with the results from previous frames (accumulated in the reprojection cache) to calculate the soft shadow for each pixel.
- When a pixel becomes newly visible and therefore no previous information is available in the reprojection cache, we use a fast single sample approach (PCSS with a fixed 4x4 kernel) to generate an initial soft shadow estimation for this pixel.
- To avoid discontinuities between sampled and estimated soft shadows, all the estimated pixels are augmented by using a depth-aware spatial filter to take their neighborhood in the *shadow buffer* into account.

This results in a very fast soft shadow approach based on shadow maps that uses temporal reprojection for converging to the physical correct result (see also Figure 10).

IV. Conclusions

This paper has given a short introduction into the field of TC in real-time rendering. TC is of course also used in many other areas of real-time rendering.

Yang et al. [23] use it for amortizing supersampling: They maintain several samples from previous frames and combine them in the current frame using reprojection. In the majority of cases they can thereby avoid the computational cost of calculating multiple samples for each fragment.

Also analysis papers of the reprojection cache approach exist. Sitthi-Amorn et al. [17] analyse the potential performance gain achievable by introducing the reprojection cache. They find that a 3-pass algorithm (in contrast to the single pass or 2-pass algorithms used before) is more efficient to execute on current hardware. The problem when and how to update the *reprojection cache* (refresh policy) is investigated in [18]. They present automatic methods to select when and which samples to refresh using a parametric model that describes the way possible caching decisions affect the visual fidelity and the shader's performance.

But due to place constrains many interesting algorithms had to be omitted. This area is a very active field of research and there are still many algorithms in real-time rendering that could also benefit from accounting for TC in the algorithm design.

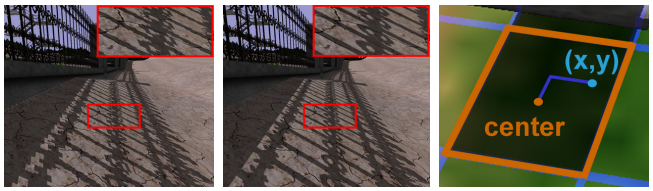

Fig. 8. LiSPSM *(left)* gives good results for a shadow map resolution of 1024^2 and a viewport of 1680×1050, but temporal reprojection *(middle)* can still give superior results because it uses shadow test confidence *(right)*: $\text{conf}_{x,y} = 1 - \max\left(|x - \text{center}_x|, |y - \text{center}_y|\right) \cdot 2$.

References

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

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

[3] Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen J. Scher Zagier. Frameless rendering: double buffering considered harmful. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 175–176, New York, NY, USA, 1994. ACM.

[4] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, New York, NY, USA, 1993. ACM.

[5] Markus Giegl and Michael Wimmer. Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. *Computer Graphics Forum*, 26(1):46–49, March 2006.

[6] Meister Eduard Gröller. *Coherence in Computer Graphics*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1992.

[7] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, CS Dept., Carnegie Mellon U., Jan. 1997. CMU-CS-97-104, http://www.cs.cmu.edu/ ph.

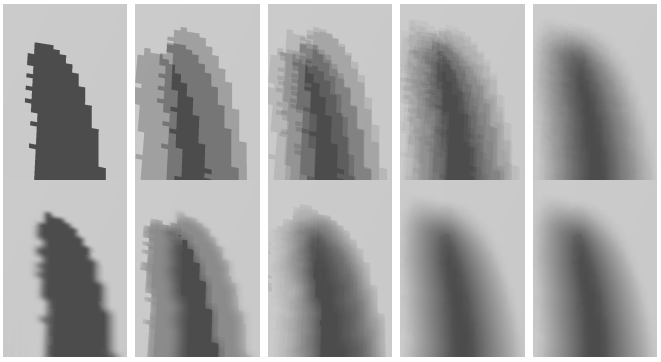

Fig. 9. Convergence after 1,3,7,20 and 256 frames. *Upper Row*: Sampling of the light source one sample per frame; *Lower Row*: Our new algorithm.

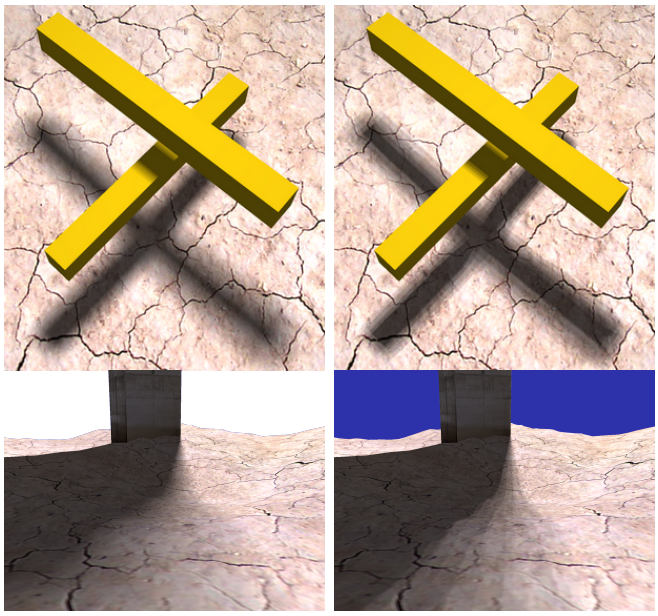

Fig. 10. *Left side*: Our Method. *Right Side*: PCSS 16/16. Overlapping occluders (upper row) and bands in big penumbras (lower row) are known problem cases for single sample approaches.

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

[9] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff., New York, NY, USA, 1997. ACM.

[10] Leonard McMillan and Gary Bishop. Head-tracked stereoscopic display using image warping. In *Proceedings SPIE, volume 2409*, pages 21–30, 1995.

[11] Diego Nehab, Pedro V. Sander, and John R. Isidoro. The real-time reprojection cache. In *ACM SIGGRAPH Sketch*, page 185, 2006.

[12] Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware*, pages 25–35, 2007.

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

[14] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Eurographics Symposium on Rendering*, pages 45–50, 2007.

[15] Daniel Scherzer, Michael Schwärzler, Oliver Mattausch, and Michael Wimmer. Real-time soft shadows using temporal coherence. *Lecture Notes in Computer Science (LNCS)*, November 2009.

[16] Daniel Scherzer and Michael Wimmer. Frame sequential interpolation for discrete level-of-detail rendering. *Computer Graphics Forum (Proceedings EGSR 2008)*, 27(4):1175–1181, June 2008.

[17] Pitchaya Sitthi-amorn, Jason Lawrence, Lei Yang, Pedro V. Sander, and Diego Nehab. An improved shading cache for modern GPUs. In *Proc. of Graphics Hardware*, pages 95–101, 6 2008.

[18] Pitchaya Sitthi-amorn, Jason Lawrence, Lei Yang, Pedro V. Sander, Diego Nehab, and Jiahe Xi. Automated reprojection-based pixel shader optimization. *ACM Trans. Graph.*, 27(5):127, 12 2008.

[19] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Comput. Surv.*, 6(1):1–55, 1974.

[20] Edgar Velázquez-Armendáriz, Eugene Lee, Kavita Bala, and Bruce Walter. Implementing the render cache and the edge-and-point image on graphics hardware. In *GI '06: Proceedings of Graphics Interface 2006*, pages 211–217, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[21] Bruce Walter, George Drettakis, and Donald P. Greenberg. Enhancing and optimizing the render cache. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 37–42, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[22] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In D. Lischinski and G.W. Larson, editors, *Rendering techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)*, volume 10, pages 235–246, New York, NY, Jun 1999. Springer-Verlag/Wien.

[23] Lei Yang, Diego Nehab, Pedro V. Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. Amortized supersampling. *ACM Trans. Graph.*, 28(5):135, 2009.

[24] Tenghui Zhu, Rui Wang, and David Luebke. A gpu-accelerated render cache. *Pacific Graphics, (Short Paper Session)*, October 2005.