NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
NWU-CS-05-07
April 26, 2005

Adaptive Frameless Rendering

Abhinav Dayal[1], Cliff Woolley[2], Benjamin Watson[1] and David Luebke[2]
[1]Northwestern University, [2]University of Virginia

Abstract

We propose an adaptive form of frameless rendering with the potential to dramatically increase rendering speed over conventional interactive rendering approaches. Without the rigid sampling patterns of framed renderers, sampling and reconstruction can adapt with very fine granularity to spatio-temporal color change. A sampler uses closed-loop feedback to guide sampling toward edges or motion in the image. Temporally deep buffers store all the samples created over a short time interval for use in reconstruction and as sampler feedback. GPU-based reconstruction responds both to sampling density and space-time color gradients. Where the displayed scene is static, spatial color change dominates and older samples are given significant weight in reconstruction, resulting in sharper and eventually antialiased images. Where the scene is dynamic, more recent samples are emphasized, resulting in less sharp but more up-to-date images. We also use sample reprojection to improve reconstruction and guide sampling toward occlusion edges, undersampled regions, and specular highlights. In simulation our frameless renderer requires an order of magnitude fewer samples than traditional rendering of similar visual quality (as measured by RMS error), while introducing overhead amounting to 15% of computation time.

Keywords: I.3.3 [Computer Graphics]: Picture-Image Generation—Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics And Realism—Raytracing; Virtual reality.

Technical Report NWU-CS-05-07
Northwestern University, Evanston, IL 60201

Adaptive Frameless Rendering

Abhinav Dayal[1], Cliff Woolley[2], Benjamin Watson[1] and David Luebke[2]
[1]Northwestern University, [2]University of Virginia

Abstract
We propose an adaptive form of frameless rendering with the potential to dramatically increase rendering speed over conventional interactive rendering approaches. Without the rigid sampling patterns of framed renderers, sampling and reconstruction can adapt with very fine granularity to spatio-temporal color change. A sampler uses closed-loop feedback to guide sampling toward edges or motion in the image. Temporally deep buffers store all the samples created over a short time interval for use in reconstruction and as sampler feedback. GPU-based reconstruction responds both to sampling density and space-time color gradients. Where the displayed scene is static, spatial color change dominates and older samples are given significant weight in reconstruction, resulting in sharper and eventually antialiased images. Where the scene is dynamic, more recent samples are emphasized, resulting in less sharp but more up-to-date images. We also use sample reprojection to improve reconstruction and guide sampling toward occlusion edges, undersampled regions, and specular highlights. In simulation our frameless renderer requires an order of magnitude fewer samples than traditional rendering of similar visual quality (as measured by RMS error), while introducing overhead amounting to 15% of computation time.

Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture-Image Generation—Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics And Realism—Raytracing; Virtual reality

1. Improving Interactive Rendering

In recent years a number of traditionally offline rendering algorithms have become interactive or nearly so. The introduction of programmable high-precision graphics processors (GPUs) has drastically expanded the range of algorithms that can be employed in real-time graphics; meanwhile, the steady progress of Moore's Law has made techniques such as ray tracing, long considered a slow algorithm suited only for offline realistic rendering, feasible in real-time rendering settings [WDB*03]. These trends are related; indeed, some of the most promising interactive global illumination research performs algorithms such as ray tracing and photon mapping directly on the GPU [PBMH02]. Future hardware should provide even better support for these algorithms, bringing us closer to the day when ray-based algorithms are an accepted and powerful component of every interactive rendering system.

What makes interactive ray tracing attractive? Researchers in the area have commented on ray tracing's ability to model physically accurate global illumination phenomena, its easy applicability to different shaders and primitives, and its output-sensitive running time, which is only weakly dependent on scene complexity [WPS*03]. We focus on another unique capability: selective sampling of the image plane. By design, depth-buffered rasterization must generate an entire image at a given time, but ray-tracing can focus rendering with very fine granularity. This ability enables a new approach to rendering that is both more interactive and more accurate.

The topic of sampling in ray tracing may seem nearly ex-

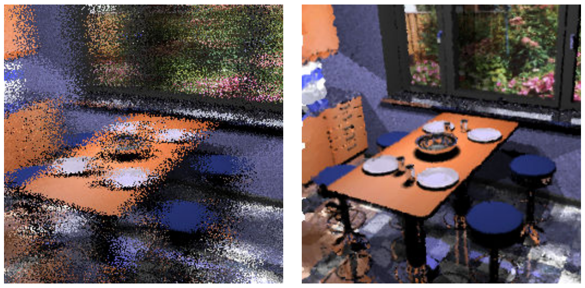
Figure 1: *Adaptive frameless rendering improves upon frameless rendering [BFMS94] (left) with adaptive sampling and reconstruction (right). Resulting imagery has similar visual quality to a framed renderer but is produced using an order of magnitude fewer samples per second.*

hausted, but almost all previous work has focused on *spatial sampling*, or where to sample in the image plane. In an interactive setting, the question of *temporal sampling*, or when to sample with respect to user input, becomes equally important. Temporal sampling in traditional graphics is bound to the frame: an image is begun in the back buffer incorporating the latest user input, but by the time the frame is swapped to the front buffer for display, the image reflects stale input. To mitigate this, interactive rendering systems increase the frame rate by reducing the complexity of the scene, trading off fidelity for performance.

In this paper we investigate novel sampling schemes for managing the fidelity-performance tradeoff. Our approach has two important implications. First, we advocate *adaptive temporal sampling*, analogous to the adaptive spatial sampling long employed in progressive ray tracing [BFGS86; M87; PS89]. Just as spatially adaptive renderers display detail *where* it is most important, temporally adaptive sampling displays detail *when* it is most important. Second, we advocate *frameless rendering* [BFMS94], in which samples are located freely in space-time rather than placed at regular temporal intervals forming frames, and with images reconstructed from a sampled space-time volume, rather than a coherent temporal slice. Frameless rendering decouples spatial and temporal sampling, enabling adaptive spatial and temporal sampling.

Our prototype adaptive frameless renderer consists of four primary subsystems. An *adaptive sampler* directs rendering to image regions undergoing significant change (in space and/or time). The sampler produces a stream of samples scattered across space-time; recent samples are collected and stored in *two* temporally *deep buffers*. One of these buffers provides feedback to the sampler, while the other serves as input to an *adaptive reconstructor,* which repeatedly reconstructs the samples in its deep buffer into an image for display, adapting the reconstruction filters to local sampling density and color gradients. Where the displayed scene is static, spatial color change dominates and older samples are given significant weight in reconstruction, resulting in sharper images. Where the scene is dynamic, only more recent samples are emphasized, resulting in a less sharp but correctly up-to-date image.

We describe an interactive system built on these principles, and show in simulation that this system achieves superior rendering accuracy and responsiveness. We compare our system's imagery to the imagery that would be displayed by a hypothetical zero-delay, antialiased renderer using RMS error. Our system outperforms not only frameless sampling (Figure 1), but also equals the performance of a framed renderer sampling 10 times more quickly.

2. Related work

Bishop et al.'s frameless rendering [BFMS94] replaces the coherent, simultaneous, double-buffered update of all pixels with samples distributed stochastically in space, each representing the most current input when the sample was taken. Pixels in a frameless image therefore represent many moments in time. Resulting images are more up-to-date than double-buffered frames, but temporal incoherence causes visual artifacts in dynamic scenes.

Inspired by frameless rendering, other researchers examined the loosening of framed sampling constraints. The just in time pixels scheme [OCMB95] takes a new temporal sample for each scanline. The address recalculation pipeline [RP94] sorts objects into several layered frame buffers refreshed at different rates. The Talisman system [TK96] renders portions of the 3D scene at different rates. Ward and Simmons [WS99] and Bala et al. [BDT99] store and reuse previously rendered rays. In work that is particularly relevant here, several researchers have studied sample reprojection, which reuses samples from previous frames by repositioning them to reflect the current viewpoint. Walter et al.'s Render Cache [WDP99; WDG02] reconstructs these temporally incoherent samples using depth comparisons and filtering that span small pixel neighborhoods. New samples are guided toward regions that have not been recently sampled, are sparsely sampled, or contain temporal color discontinuities. Simmons and Séquin [SS00] use a hardware interpolated 2.5D mesh to cache and reconstruct the samples, and guide new samples toward spatial color and depth discontinuities. Tolé et al.'s Shading Cache [TPWG02] stores samples in the 3D scene itself, performing reconstruction by rendering that scene in hardware. Sampling is biased toward spatial color discontinuities and toward specular and moving objects. Havran et al. [HDM03] calculate the temporal intervals over which a given sample will remain visible in an offline animation and reproject that sample during the interval. Shading is recalculated for reprojected samples in every frame. Although the images they produce combine samples created at many different moments, all of these systems sample time at regular intervals.

Woolley et al. [WLWD03] describe a fully framed but temporally adaptive sampling scheme called interruptible rendering. The approach adaptively controls frame rate to minimize simultaneously the error created by reduced rendering fidelity and by reduced rendering performance. A progressive renderer refines a frame in the back buffer until the error created by unrepresented input exceeds the error caused by coarse rendering. At that point, the front and back buffers are swapped and rendering begins again into the back buffer using the most recent input. Coarse, high frame-rate display results when input is changing rapidly, and finely detailed, low frame rate display when input is static.

Many advances in high-speed ray tracing have been made recently. These include clever software techniques to im-

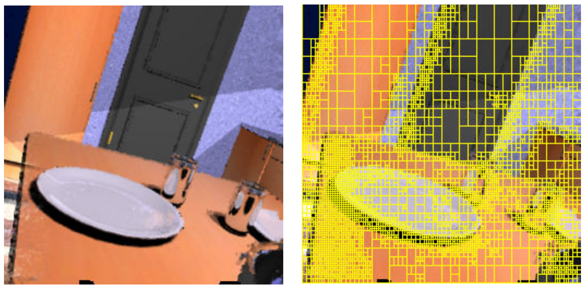
Figure 2: *A reconstructed image and an overlay showing the tiling used by the sampler at that moment in time. Note the finer tilings over object edges and occlusions.*

prove memory locality [PKGH97; TA98; WBWS01], as well as advances in hardware that enable interactive ray tracers on supercomputers [PMS*99], on PC clusters [WSB01; WBDS03], on the SIMD instruction sets of modern CPUs [WBWS01], and on graphics hardware [PBMH02; CHH02]. Wald et al. provide a good summary of the state of the art [WPS*03]. These advances will soon allow a very fine-grained and selective space-time sampling, in real time.

This real-time, selective sampling enables a new adaptive form of frameless rendering that incorporates techniques from adaptive renderers, reprojecting renderers, non-uniform reconstruction [M87], and GPU programming. The resulting system outperforms framed and traditional frameless renderers and offers the following advantages over reprojecting renderers:

Improved sampling response. Rather being clustered at each frame time, samples reflect the most up-to-date input available at the moment they are created. Further, closed-loop control guides samples toward not only spatial but temporal color discontinuities at various scales. These elements combine to reduce rendering latency.

Improved reconstruction. Rather than being non-adaptive or hardware-interpolated, reconstruction is adaptive over both space and time, responding to local space-time color gradients. This drastically improves image quality, eliminating the temporal incoherence in traditional frameless imagery without requiring framed sampling and its increased latency, and permitting antialiasing in static image regions. Moreover this reconstruction is already interactive and implemented on existing GPU hardware.

3. Adaptive frameless sampling

Previous importance sampling techniques [BFGS86; G95; M87; PS89] are spatially adaptive, focusing on regions where color changes across space. Our renderer is both spatially and temporally adaptive, focusing also on regions where color changes over time (Figure 2). Adaptive bias is added to sampling with the use of a spatial hierarchy of image-space *tiles*. However, while previous methods operated in the static context of a single frame, we operate in a dynamic frameless context. This has several implications. First, rather than operating on a frame buffer, we send samples to two temporally *deep buffers* that collect samples scattered across space-time (one buffer for the sampler, one for the reconstructor). Our tiles therefore partition a space-time volume using planes parallel to the temporal axis. We call each resulting subvolume a *block*. Second, as in framed schemes, color variation within each tile guides rendering bias, but variation represents change over not just space but also time. Moreover, variation does not monotonically decrease as the renderer increases the number of tiles, but rather constantly changes in response to user interaction and animation. Therefore the hierarchy is also constantly changing, with tiles continuously merged and split in response to dynamic changes in the contents of the deep buffer.

```
fill deep buffers non-adaptively
loop
    choose a tile to render and a pixel within it
    find last location sampled in pixel
    complete a crosshair of samples at last location
    update deep buffers and tile statistics
    repeat 5 times
        choose a tile crosshair and reproject it
        reevaluate tile gradients in crosshair
        check visibility of crosshair center sample
        if occluded then create new crosshair at same location
        update deep buffers and tile statistics
    end repeat
    choose a different pixel in tile to sample
    create sample, update last location sampled in pixel
    update deep buffers and tile statistics
    if one display time elapsed
        then send reconstructor view and tile information
    if another chunk of crosshairs has been completed
        then adjust tiling
end loop
```

Figure 3: *Pseudocode for the main loop in the sampler.*

The sampler's deep buffer provides it with important feedback. This deep buffer is a 3D array sized to match the number of image pixels in two dimensions, and a shallow buffer depth b in the third, temporal dimension (we use $b = 4$). Buffer entries at each pixel location form a queue, with new samples inserted into the front causing the removal of the sample in the back if the queue is full. Each sample is also sent to the reconstructor's buffer as soon as it arrives in the sampler's buffer, and is described by its color, position in world space, age, and a view-independent velocity vector. In addition to filling the reconstructor's deep buffer, the sampler sends the reconstructor regular updates describing the current view and tiling. This information is sent to the reconstructor 60 times per second, and includes each tile's image coordinates as well as the average temporal and spatial color gradients in the tile's block.

We implement our sampler's tiling hierarchy using a K-D tree. Given a target number of tiles, the tree is managed to

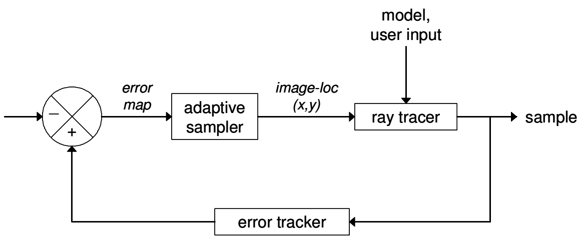

Figure 4: *Adaptive frameless sampling as closed loop control. Samples from the ray tracer (plant) are sent to an error tracker, which adjusts the tiling or error map. The adaptive sampler (compensator) then selects one location to render in a tile. Constantly changing user input (disturbance) makes it very difficult to track and limit error.*

ensure that the amount of color variation in each tile's block is roughly equal: the tile with the most color variation is split and the two tiles with the least summed variation are merged, until all tiles have roughly equal variation. We calculate variation across all of a tile's samples using the equation $v_{tile} = 1/n\ \Sigma_i\ (L_i - L_m)^2$, where L_i is a sample's luminance and L_m the mean luminance in the tile. We ensure prompt response to changes in scene content by weighting samples in the variance calculation using a function that declines exponentially as sample age increases. As a result, small tiles are located over dynamic or finely detailed buffer regions, while large tiles emerge over static or coarsely detailed regions (Figure 2). The tiling is updated after a *chunk* of c new samples has been generated (we set $c = 150$).

Sampling now becomes a biased probabilistic process (Figure 3). Since the current time is not fixed as it would be in a framed renderer, we cannot just iteratively sample the tile with the most variation—in doing so, we would overlook newly emerging motion and detail. At the same time, we cannot leave rendering unbiased and unimproved. Our solution is to select each tile with equal probability and select the sampled location within the tile using a uniform distribution. Because tiles vary in size, sampling is biased towards those regions of the image which exhibit high spatial and/or temporal variance. Because all tiles are sampled, we remain sensitive to newly emerging motion and detail.

This sampler thus constitutes a closed-loop control system [DTB97], capable of adapting to user input with great flexibility (Figure 4). In control theory, the *plant* is the process being directed by the *compensator*, which must adapt to external *disturbance*. Output from the plant becomes input for the compensator, closing the feedback loop. In a classic adaptive framed sampler, the compensator chooses the rendered location, the ray tracer is the plant that must be controlled, and disturbance is provided by the scene as viewed at the time being rendered. Our frameless sampler faces a more difficult challenge: view and scene state may change after each sample.

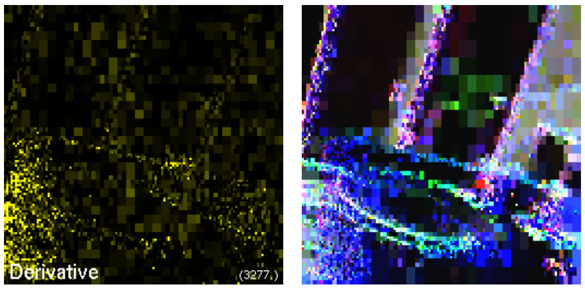

Figure 5: *Error derivatives (left) and the tile gradients (right) G_x, G_y, and G_t (shown here as red, green, and blue, respectively) in a scene corresponding to Figure 2.*

To meet this challenge, we apply two control engineering techniques. We first use a PD controller, in which control responds not only in proportion to error itself (P), but also to its derivative (D). In our sampler, error is color variation, and by biasing sampling toward variation, we are already responding in proportion to it. By responding to error's derivative, we bias sampling toward regions in which variation is changing such as the edge of the moving table in Figure 2, compensating for delay in our control system. We accomplish this by tracking variation change d and adding it to normalized variation p to form a new summed control error e in the tile:

$$e = kp + (1\text{-}k)d,$$

where

$$p = {v_{tile}\, s} \Big/ {\textstyle\sum_{j}^{|tiles|} v_j}$$

is the temporally weighted color variance in the tile block v_{tile} normalized by the sum of these variances over all tiles scaled by the tile size s, d is the absolute difference between p's current value and its value u updates of the tile ago (we use $u = 4$) divided by the time between those updates, and k in the range [0,1] is the weight applied to the proportional term. The left image in Figure 5 visualizes d for each tile by mapping high d values to brighter colors.

Our prototype adaptive sampler will be less effective when the rendered scene is more dynamic, changing the desired image (or *target signal* in control theory) more rapidly. In such cases, fixed delays in response will make control increasingly ineffective. To address this problem we apply another control engineering technique: adjusting *gain*. We implement this by restricting or increasing the ability of the sampler to adapt to deep buffer content. Specifically, we adjust the number of tiles onscreen so that color change over space and time are roughly equal in all tiles by ensuring that $dC/ds\ S = i\ dC/dt\ T$, where dC/ds and dC/dt are spatial and temporal color gradients averaged over the entire image (Figure 5), S is the average width of the tiles, T the average age of the samples in each tile, and i is a constant adjusting the relative importance of temporal and spatial change in control. By solving for S we can derive the appropriate number of tiles.

We find current spatial gradients by sampling five tightly clustered image locations (xy, $x_{\pm1}y$ and $xy_{\pm1}$) in a crosshair pattern each time we add samples to the deep buffers, and averaging the horizontal and vertical absolute differences. To find a temporal gradient, we find the absolute difference between the center *xy* sample and a sample made at the same location the last time we visited the same pixel region, and divide by the time elapsed since that previous sample was made. We then produce and store a sample at a new location in the pixel region for pairing with the next crosshair made in the pixel region. This set of six samples forms a single entry in the *xy* queue of the sampler's deep buffer (they are not grouped when sent to the reconstructor's deep buffer). To determine average tile gradients, we reduce the weight of each sample gradient as a function of time using the same exponential scheme used to track color variation.

To permit antialiasing, we center sample crosshairs at random spatial locations. However when the scene is particularly dynamic and spatial sampling density is decreased, sharp edges may appear to "shimmer" in reconstructed imagery. Although adaptive reconstruction reduces these artifacts, we eliminate them by randomizing crosshair location only when the scene is locally static and temporal gradients approach zero.

4. Interactive space-time reconstruction

Frameless sampling strategies demand a rethinking of the traditional computer graphics concept of an "image", since at any given moment the samples in an image plane represent many different moments in time. The original frameless work [BFMS94] simply displayed the most recent sample at every pixel. This *traditional reconstruction* results in a noisy image that appears to sparkle when the scene is dynamic (see Figure 1). In contrast, we convolve the frameless samples in the reconstructor's temporally deep buffer with space-time filters to continuously reconstruct images for display. This is similar to the classic computer graphics problem of reconstruction of an image from non-uniform samples [M87], but with a temporal element: since older samples may represent "stale" data, they are treated with less confidence and contribute less to nearby pixels than more recent samples. The resulting images greatly improve over traditional reconstruction (see again Figure 1).

4.1. Choosing a filter

The key question is what shape and size filter to use. A temporally narrow, spatially broad filter (i.e. a filter which falls off rapidly in time but gradually in space) will give very little weight to relatively old samples, emphasizing the newest samples and leading to a blurry but very current image. Such a filter provides low-latency response to changes and should be used when the underlying image is changing rapidly. A temporally broad, spatially narrow filter will give nearly as much weight to relatively old samples as to recent samples; such a filter accumulates the results of many samples and leads to a finely detailed, antialiased image when the underlying scene is changing slowly. However, often different regions of an image change at different rates, as for example in a stationary view in which an object is moving across a static background. A scene such as this demands spatially adaptive reconstruction, in which the filter extent varies across the image. What should guide this process?

We use local sampling density (Figure 7) and space-time gradient information (Figure 5) to guide filter size. The reconstructor maintains an estimate of local sampling density across the image, based on the overall sampling rate and on the tiling used to guide sampling. We size our filter support—which can be interpreted as a space-time volume—as if we were reconstructing a regular sampling with this local sampling density, and while preserving the total volume of the filter, perturb the spatial and temporal filter extents according to local gradient information. A large spatial gradient implies an edge, which should be resolved with a narrow filter to avoid blurring across that edge. Similarly, a large temporal gradient implies a "temporal edge" such as an occlusion event, which should be resolved with a narrow filter to avoid including stale samples from before the event. This is equivalent to an "implicit" robust estimator; rather than searching for edges explicitly, we rely on the gradient to allow us to size the filter such that the expected contribution of samples past those edges is small.

Thus, given a local sampling rate R_l, expressed in samples per pixel per second, we define V_S as the expected space-time volume occupied by a single sample:

$$V_S = \frac{1}{R_l} .$$

The units of V_S are pixel-seconds per sample (note that the product of pixel areas and seconds is a volume). We then construct a filter at this location with space-time support proportional to this volume. For simplicity we restrict the filter shape to be axis-aligned to the spatial *x* and *y* and the temporal *t* dimensions. The filter extents e_x, e_y, and e_t are chosen to span equal expected color change in each dimension, determined by our estimates of the gradients G_x, G_y, and G_t and the total volume constraint V_s:

$$e_x G_x = e_y G_y = e_t G_t$$

$$V_S = e_x e_y e_z .$$

Thus the filter extents are given by

$$e_x = \sqrt[3]{\frac{V_S G_y G_t}{G_x^2}}, e_y = \sqrt[3]{\frac{V_S G_x G_t}{G_y^2}}, e_t = \sqrt[3]{\frac{V_S G_x G_y}{G_t^2}} .$$

What function to use for the filter kernel remains an open question. According to signal theory, a regularly sampled, band limited function should be reconstructed with a sinc function, but our deep buffer is far from regularly sampled

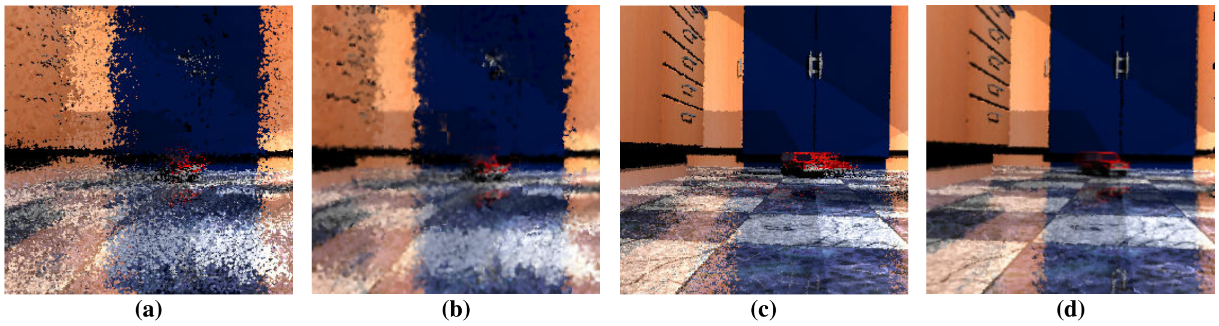

Figure 6: *Adaptive reconstruction illustrated in one moment of a scene with a moving view and car, sampled using our adaptive frameless techniques. In (a), traditional frameless reconstruction leaves many artifacts of the view motion in the image. In (b), adaptive reconstruction rejects many of the outdated samples, eliminating artifacts and clarifying edges. (c) shows the improvements possible by reprojecting samples as in [WDP99], even without adaptive reconstruction. When reprojection is combined with adaptive reconstruction as in (d), the car's motion and view-dependent reflections in the floor are clarified.*

and the underlying signal (an image of a three-dimensional scene) contains high-frequency discontinuities such as occlusion boundaries. We have experimented with a range of filters. Box and tent filters have poor bandpass properties but are extremely cheap to evaluate. A gaussian filter looks better but also requires more computation. The Mitchell-Netravali filter [M87] is considered among the best filters for nonuniform sampling, but is still more costly and requires more precision than is provided by our 16-bit GPU implementation. We have also experimented with a simple inverse exponential filter, which has the nice temporal property that the relative contribution of two samples does not change as both grow older; however, the bandpass properties of this filter are less than ideal. We are currently using a gaussian filter.

4.2. Scatter versus gather

We can consider reconstruction a *gather* process which loops over the pixels, looks for samples in the neighborhood of each pixel, and evaluates the contribution of those samples to that pixel. Alternatively, we can cast reconstruction as a *scatter* process which loops over the samples, projects each onto the image plane, and evaluates its contribution to all pixels within some footprint. We have experimented with both approaches.

We implemented the reconstructor initially as a gather process directly on the sampler's deep buffer. At display time the reconstructor looped over the pixels, then adjusted the filter size and extents at each pixel using gradient and local sample density as described above. The reconstructor gathered samples outwards from each pixel in space and time until the maximum possible incremental contribution of additional samples would be less than some threshold ε. The final color at that pixel was computed as the normalized weighted average of sample colors. This process proved expensive in practice—our unoptimized simulator required reconstruction times of several hundred ms for small (256×256) image sizes. It was also unclear how to efficiently implement hardware sample reprojection

We have therefore moved to a scatter-based implementation that stores the N most recent samples produced by the sampler across the entire image; the value of N is typically at least 4× the desired image resolution. This store is a distinct deep buffer for the reconstructor that organizes the samples as a single temporally ordered queue rather than a spatial array of crosshairs. At reconstruction time, the system splats each of these samples onto the image plane and evaluates the sample's affect on every pixel within the splat extent by computing the distance from the sample to the pixel center and weighting the sample's color contribution according to the local filter function. These accumulated contributions are then divided by the accumulated weight at each pixel to produce the final image (Figure 6).

We implement this scatter approach on the GPU, achieving real-time or near real-time performance and improving on the speed of our CPU-based gather implementation by almost two orders of magnitude. The GPU treats the samples in the deep buffer as vertices in a vertex array, and uses an OpenGL vertex program to project them onto the screen as splats (i.e., large GL_POINTS primitives). A fragment program runs at each pixel covered by a sample splat, finding the distance to the sample and computing the local filter shape by accessing tile information—local filter extent, precomputed from sample density and G_x,G_y,G_t gradients—stored in a texture. This texture is periodically updated by rasterizing the latest tiling (provided by the sampler) as a set of rectangles into an offscreen buffer. To reduce overdraw while still providing broad filter support in sparsely sampled regions, the vertex program rendering the samples adaptively adjusts point size. Section 5.2 describes this process in more detail.

The reconstructor uses several features of recent graphics hardware, including floating-point textures with blend support, multiple render targets, vertex texture fetch, dynamic branching in vertex programs, and separate blend functions for color and alpha. The results presented in this paper were obtained on a NVIDIA GeForce 6800 Ultra. In general, reconstruction occurs at 20 Hz rates when keeping a visually sufficient number of samples *N* (*N*=400K). This is remarkable considering that our use of the hardware differs greatly from the rendering task the hardware was designed to support.

5. Reprojection

Our adaptive frameless sampling and reconstruction techniques operate entirely in 2D image space and do not rely on information about sample depth or the 3D structure of the scene. However, because camera location and sample depth are easily available from our ray-tracing renderer, we also incorporate sample reprojection [WDP99; WDG02; BDT99; WS99] into our algorithms. During sampling, reprojection can help the sampler find and focus on image regions undergoing disocclusion, occlusion, view-dependent lighting changes, or view-independent motion. During reconstruction, sample reprojection extends the effective "lifetime" of a sample by allowing older samples to contribute usefully to imagery even after significant camera or object motion. This section describes our strategies for using reprojection with our sampler and reconstructor.

5.1. Reprojection in the sampler

It is not necessary to reproject every sample at fixed intervals, and indeed this would not be desirable since it would introduce periodicity into our frameless sampler. Instead, we reproject a small number of recent samples as we generate each new sample. When updates of tiling statistics (e.g. variation, gradients) are included, reprojecting a sample takes roughly 1/35th the mean time required to generate a new sample. We therefore reproject a small number (currently 5) of a tile's crosshairs each time the sampler visits a tile. In this way the same useful rendering bias that guides generation of new samples determines which samples are reprojected, focusing reprojections on important image areas.

Within each tile, we choose the corresponding pixels to reproject randomly and relocate the crosshairs from the front of each pixel's queue in the deep buffer. We apply both motion and viewing transformations to the samples in the crosshair. Despite being updated in some sense, reprojected samples continue to age normally and do not receive increased weight in variance and gradient calculations. On a local per-tile basis, every sample is treated similarly, and its age remains a good indicator of its utility. We determine a crosshair's new location in the buffer solely by its relocated center sample, and insert the crosshair sample at the back of its new queue, updating source and destination tile statistics if necessary. When updating tile gradients, spatial gradients for the crosshair are recalculated using the new spatial locations of the crosshair samples (after reprojection, they are no longer separated by one pixel length). We recalculate the crosshair temporal gradients by finding the absolute difference between the reprojected center sample and the newest sample in that pixel region, and dividing this difference by the age of this newest sample. Reprojection of a crosshair continues until it ages so much that it no longer affects tile variance and gradients, or until it is pushed out of its queue in the deep buffer by a newly arriving crosshair.

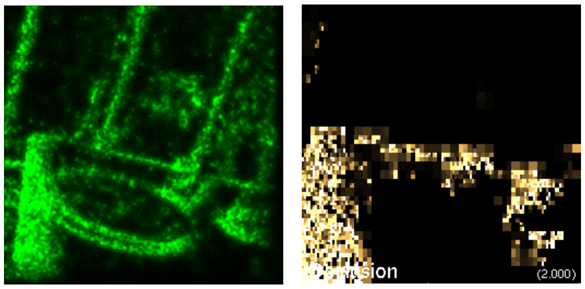

Figure 7: *A sample density map (left) used by the reconstructor to determine the expected local sample volume V_s, and occlusion detection (right) used direct sampling.*

Regions containing disocclusions will be undersampled as samples reproject to other image locations. We bias sampling toward these disocclusions with a new undersampling measure u_{tile}:

$$u_{tile} = 1 - \min\left(1, \frac{m\sum_{j=1}^{|tiles|}(whb - |buffer|) \big/ |tiles|}{sb - |tile|}\right).$$

Here the number of empty samples in a tile must be *m* times greater than the mean number of empty samples in all tiles to affect sampling. |*buffer*| and |*tile*| are the number of samples in the deep buffer and the current tile's block, while *whb* is the number of samples the deep buffer can hold (with image size $w \times h$).

Regions undergoing occlusion will contain samples from multiple surfaces at differing view depths, leading to uncertainty about image content. To resolve this uncertainty, we increase sampling in occluded regions. We detect occlusions by casting rays from the eye to each reprojected sample. If the sample is no longer visible from the eye, we add a new sample at the occluded image location. We also increase sampling density in the occluded region by increasing error in tiles experiencing occlusion with an occlusion term o_{tile} = |*O*|/*sb*, where |*O*| is the number of occluded samples in a tile's block, tracked by our occlusion test. Figure 7 shows the occlusions that affect sampling. Proportional error *p* for the sampler then becomes:

$$p = s\left(\kappa^{v_{tile}} \big/ \sum_j^{|tiles|} v_j + \lambda^{u_{tile}} \big/ \sum_j^{|tiles|} u_j + (1-\kappa-\lambda)^{o_{tile}} \big/ \sum_j^{|tiles|} o_j\right) \text{w}$$

ith κ, λ, and (κ + λ) all in [0,1].

5.2. Reprojection in the reconstructor

Unlike the sampler, the reconstructor operates in a framed context: to display an image on existing hardware, it scans out a traditional image (i.e., a uniform grid of pixels) at the regular intervals of the display refresh. Since each sample in the reconstructor's deep buffer stores the 3D hit point of the primary ray that generated that sample, reprojecting and reconstructing each of our renderer's images reduces to rendering the vertex array in the deep buffer with the current camera and projection matrices bound. Figure 6 shows the results of using reprojection in reconstruction.

Reprojection sometimes generates regions of low sample density, for example at disocclusions and near the leading edge of the screen during camera rotation. In such regions, the filter support for the few samples present must be quite large, requiring the reconstructor to rasterize samples with large splats. Rather than rasterizing all samples using large splats, we avoid overdraw with an adaptive point size scheme. All samples are accumulated into a *coverage map* during rendering that tracks the number and average splat size of all samples rendered to each pixel. To size splats, the sample vertex program binds the previous image's coverage map as a texture, computes the projected coordinates of the sample, and uses the coverage information at those coordinates to calculate the splat size at which the sample will be rasterized. Sample splats in a region are sized according to the average point size used in that region during reconstruction of the previous image, but point sizes in undersampled regions (defined currently as fewer than 4 samples affecting a pixel) are multiplied by 4 to grow rapidly, while point sizes in oversampled regions (more than 32 samples reaching a pixel) are multiplied by 0.7 to shrink gradually.

6. Evaluation

Using the *gold standard* validation described in [WLWD03], we find that our adaptive frameless renderer consistently outperforms other renderers that have the same sampling rates.

Gold standard validation uses as its standard an *ideal renderer I* capable of rendering antialiased imagery in zero time. To perform comparisons to this standard, we create *n* ideal images I_j (j in $[1,n]$) at 60 Hz for a certain animation A using a simulated ideal renderer. We then create *n* more images R_j for animation A using an actual interactive renderer R. We next compare each image pair (I_j,R_j) using an image comparison metric *comp*. Here we use root-mean-squared error (RMS).

We report the results of our gold standard evaluation in Table 1, which compares several rendering methods producing 256x256 images using various sampling rates. Two framed renderings either maximize Hz at the cost of spatial resolution (lo-res), or spatial resolution at the cost of Hz (hi-res).

Render Method	Animation/Sampling rate						
	Interactive			Bart		Toycar	
	100k	400k	800k	400k	800k	400k	800k
Framed: lo-res	92.7	71.8	60.9	112	100	47	42.8
Framed: hi-res	110	72.6	60.4	127	112	43.8	38.9
Traditional Frameless	80.8	48.8	39.3	92.3	74.8	35.3	32.5
Adaptive	34.4	24.1	23.6	50.1	51.9	20.4	18.5
hi-res 60Hz	28	28	28	30.7	30.7	29.4	29.4

Table 1: *Summary error analysis using the techniques of Figure 8, with some additional sampling rates.*

The traditional frameless rendering simply displays the newest sample at a given pixel. The adaptive frameless renderings use our system to produce the imagery. The hi-res 60Hz is a framed renderer that uses a sampling rate 10 times higher than other renderers to produce full resolution imagery at 60Hz. (The difference between the ideal renderer and this hi-res 60Hz renderer is that the latter suffers from double-buffering delay and does not use anti-aliasing). Rendering methods were tested in 3 different animations, all using the publicly available BART testbed [LAM00]: the viewpoint animation in the testbed itself (BART); a fixed viewpoint close-up of a moving car (toycar), and a recording of user viewpoint interaction (interactive).

Adaptive frameless rendering is the clear winner, with lower RMS error than all techniques using the same sampling rate and comparable error to the hi-res 60Hz rendering, which uses sampling rates 40, 10 and 5 times faster than the 100K, 400K and 800K adaptive frameless renderings.

Figure 8 offers a more detailed view that confirms this impression. The graphs here show frame-by-frame RMS error comparisons between several of these rendering techniques and the ideal rendering. Note the sawtooth pattern produced by the low-sampling rate hi-res renderer, due to double buffering error. In the interactive animation, the periodic increases in error correspond to periods of viewpoint change. Once more, adaptive frameless rendering has lower RMS error than all rendering techniques using equivalent sampling rates, and comparable error to the much more densely sampled hi res 60Hz renderer. The top right graph also depicts the advantage of using reprojection in the sampler. RMS error is considerably higher if reprojection is not used.

7. Discussion and future work

Frameless rendering and selective sampling have been criticized for sacrificing spatial coherence and thus memory locality, which can reduce sampling speed. We plan to experiment with increases in the number of samples we generate each time we visit a tile, increasing spatial coherence at the cost of slightly less adaptive sampling overall. However, exploiting spatial coherence has its limits: ultimately, it will limit our ability to take advantage of temporal coherence and force us to sample more often. Traditional renderers must sample every single pixel dozens of times each second; as displays grow in size and resolution, this ceaseless sampling

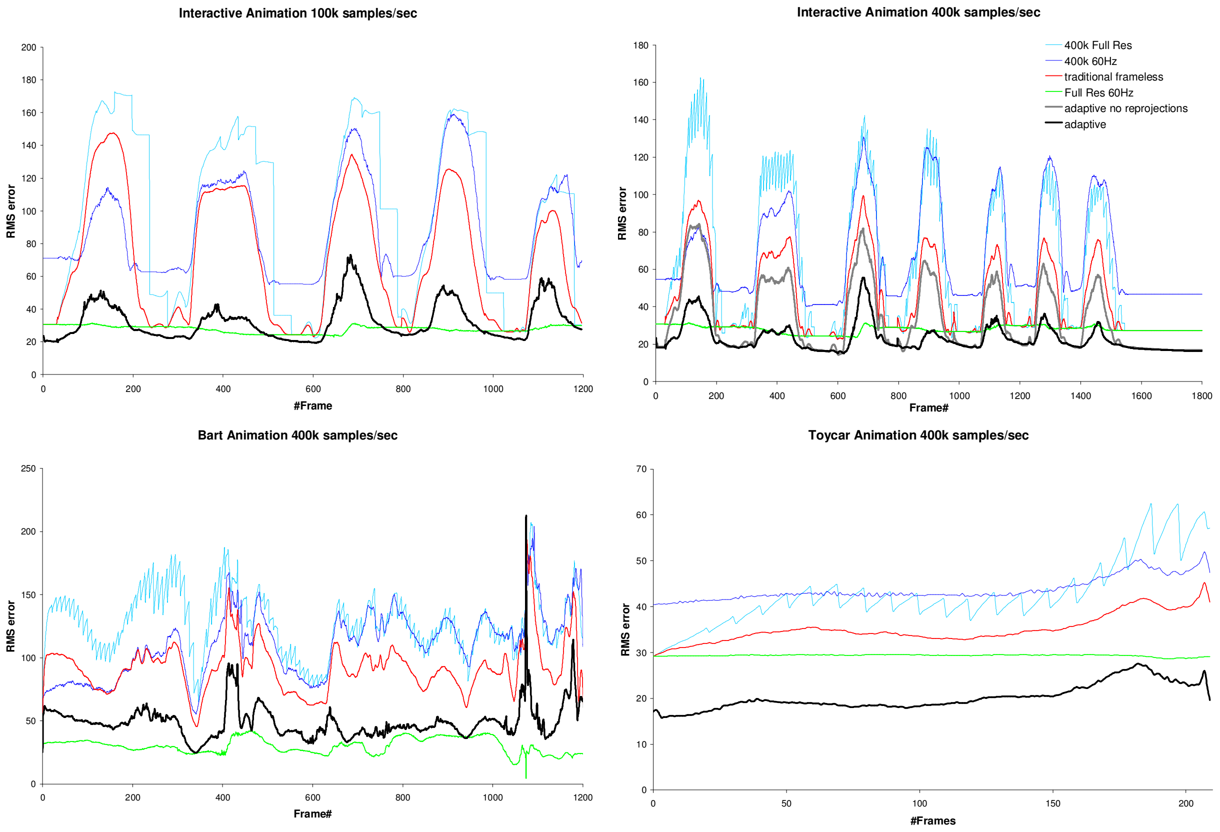

Figure 8: *An error analysis of rendering techniques for several animation sequences created using 100K or 400K samples/sec. Graphs show frame-by-frame RMS error between each technique's images and the ideal image that would be displayed by a hypothetical zero-delay, antialiased renderer at the same moment in time. Resolution is 256x256 pixels at 60 Hz.*

becomes wasteful of computation, power, and heat. With this work, we hope to shift the emphasis of interactive ray tracing research from spatial to temporal coherence, and from brute-force to selective sampling.

Good filter design for the adaptive space-time reconstruction of frameless sample streams remains an open problem. We have begun investigating the edge-preserving concepts of bilateral and trilateral filtering [DD02; CT03], which perform nonlinear filtering by weighting samples according to their difference in luminance as well as their distance in space. However, extending these approaches to include a third temporal dimension and to operate on a non-uniformly distributed samples presents a significant challenge. A related possibility is to exploit a priori information about the underlying model or animation, as do Bala et al. [BWG03].

We believe this work has great potential, and will continue this research in several longer-term directions. Extending our temporally adaptive methods to more sophisticated global illumination algorithms is one obvious avenue. With its ability to selectively alter sampling and reconstruction across both space and time, our adaptive frameless renderer is an ideal platform for experimenting with perceptually driven rendering in interactive settings [LRC02]. We are studying the possibility of extremely high resolution ("gigapixel") display hardware fed streams of frameless samples, with adaptive reconstruction performed in the display itself. This might be one solution to the immense bandwidth challenge posed by such displays. Such a rendering configuration would also enable a truly asynchronous parallelism in graphics, since renderers would no longer have to combine their samples into a single frame [MCEF94]. For this reason we are particularly interested in implementing these algorithms in graphics hardware.

8. Conclusion

In conclusion, we advocate a revival of frameless rendering, based on temporally adaptive sampling and reconstruction, and enabled by recent advances in interactive ray tracing. This approach improves traditional framed and frameless rendering by focusing sampling on regions of spatial and temporal change, and with adaptive reconstruction that emphasizes new samples when scene content is changing quickly and incorporates older samples when the scene is static. In testing, our prototype system displays greater accuracy than framed and frameless rendering schemes at comparable sampling rates, and comparable accuracy to a framed renderer sampling 10 times more quickly. Based on these results, we believe that a temporally adaptive frameless approach shows great promise for future rendering algorithms and hardware.

9. Acknowledgements

We would like to thank Edward Colgate and Kevin Lynch (Department of Mechanical Engineering, Northwestern University) for their fruitful discussions on control systems and the Saarland University Graphics Group for providing us with the OpenRT ray tracer. This work was supported in part by the NSF grants 0092973, 0093172, 0112937, and 0130869. The 3D model (kitchen) is the courtesy of the BART (A Benchmark for Animated Ray Tracing) project at Chalmers University of Technology, Sweden.

10. References

[BDT99] BALA, K., DORSEY, J., TELLER, S. 1999. Radiance interpolants for accelerated bounded-error ray tracing. *ACM Trans. Graph*, 18, 3, 213-256.

[BWG03] BALA, K., WALTER, B., GREENBERG, D.P. 2003. Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph., 22*, 3, 631–640 (*Proc. ACM SIGGRAPH*).

[BFGS86] BERGMAN, L., FUCHS, H., GRANT, E., SPACH, E. 1986. Image rendering by adaptive refinement. *Proc. ACM SIGGRAPH*, 29–37.

[BFMS94] BISHOP, G., FUCHS, H., MCMILLAN, H., SCHER ZAGIER, E.J. 1994. Frameless rendering: double buffering considered harmful. *Proc. ACM SIGGRAPH*, 175–176.

[CHH02] CARR, N.A., HALL, J.D., HART, J.C. 2002. The ray engine. *Proc. ACM SIGGRAPH/Eurographics Graphics Hardware*, 37–46.

[CT03] CHOUDHURY, P., TUMBLIN, J. 2003. The trilateral filter for high contrast images and meshes. *Proc. Eurographics Workshop on Rendering*, 186–196.

[DD02] DURAND, F., DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graphics, 21*, 3, 257–266 (*Proc. ACM SIGGRAPH*).

[DTB97] DUTTON, K., THOMPSON, S., BARRACHLOUGH, B. 1997. *The Art of Control Engineering, 1st ed.* Addison-Wesley.

[G95] GLASSNER, A. 1995. *Principles of Digital Image Synthesis, 1st ed.* Morgan Kaufmann.

[HDM03] HAVRAN, V., DAMEZ, C., MYSZKOWSKI, K. 2003. An efficient spatio-temporal architecture for animation rendering. *Proc. Eurographics Symposium on Rendering*, 106-117.

[J01] JENSEN, H.W. 2001. *Realistic Image Synthesis Using Photon Mapping*. AK Peters.

[LAM00] LEXT, J., ASSARSSON, U., MOELLER, T. 2000. Bart: A benchmark for animated ray tracing. Tech. Rpt. 00-14, Dept. Computer Engineering, Chalmers Univ. Tech. http://www.ce.chalmers.se/BART.

[LRC*02] LUEBKE, D., REDDY, M., COHEN, J.D., VARSHNEY, A., WATSON, B., HUEBNER, R. 2002. *Level of Detail for 3D Graphics, 1st ed.* Morgan Kaufmann.

[M87] MITCHELL, D.P. 1987. Generating antialiased images at low sampling densities. *Proc. ACM SIGGRAPH*, 65–72.

[MCEF94] MOLNAR, S., COX, M., ELLSWORTH, D., FUCHS, H. 1994. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, *14*, 4, 23–32.

[OCMB95] OLANO, M., COHEN, J., MINE, M., BISHOP, G. 1995. Combatting rendering latency. *Proc. ACM Interactive 3D Graphics*, 19–24.

[PS89] PAINTER, J., SLOAN, K. 1989. Antialiased ray tracing by adaptive progressive refinement. *Proc. ACM SIGGRAPH*, 281–288.

[PMS*99] PARKER, S., MARTIN, W., SLOAN, P.-P.J., SHIRLEY, P., SMITS, B., HANSEN, C. 1999. Interactive ray tracing. *Proc. ACM Interactive 3D Graphics*, 119–126.

[PKGH97] PHARR, M., KOLB, C., GERSHBEIN, R., HANRAHAN, P. 1997. Rendering Complex Scenes with memory-coherent ray tracing. *Proc. ACM SIGGRAPH*, 101– 108.

[PBMH02] PURCELL, T.J., BUCK, I., MARK, W.R., HANRAHAN, P. 2002. Ray tracing on programmable graphics hardware. *ACM Trans. Graphics, 21*, 3, 703–712 (*Proc. ACM SIGGRAPH*).

[RP94] REGAN, M.J.P., POSE, R. 1994. Priority rendering with a virtual reality address recalculation pipeline. *Proc. ACM SIGGRAPH*, 155–162.

[SS00] SIMMONS, M., SÉQUIN, C. 2000. Tapestry: A dynamic mesh-based display representation for interactive rendering. *Proc. Eurographics Workshop on Rendering*, 329–340.

[TA98] TELLER, S., ALEX, J. 1998. Frustum Casting for Progressive, Interactive Rendering. *Massachusetts Institute of Technology Technical Report LCS TR-740.* Available at http://graphics.csail.mit.edu/pubs/MIT-LCS-TR-740.ps.gz

[TPWG02] TOLE, P., PELLACINI, F., WALTER, B., GREENBERG, D.P. 2002. Interactive global illumination in dynamic scenes. *ACM Trans. Graphics, 21*, 3, 537–546 (*Proc. ACM SIGGRAPH*).

[TK96] TORBORG, J., KAJIYA, J. 1996. Talisman: Commodity Reality Graphics for the PC. *Proc. ACM SIGGRAPH*, 353-363.

[WBDS03] WALD, I., BENTHIN, C., DIETRICH, A., SLUSALLEK, P. 2003. Interactive distributed ray tracing on commodity PC clusters—state of the art and practical applications. *Lecture Notes on Computer Science*, *2790*, 499–508 (*Proc. EuroPar*).

[WBWS01] WALD, I., BENTHIN, C., WAGNER, M., SLUSALLEK, P. 2001. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, *20*, 153–164 (*Proc. Eurographics*).

[WPS*03] WALD, I., PURCELL, T.J., SCHMITTLER, J., BENTHIN, C., SLUSALLEK, P. 2003. Realtime ray tracing and its use for interactive global illumination. *Eurographics State of the Art Reports*.

[WSB01] WALD, I., SLUSALLEK, P., BENTHIN, C. 2001. Interactive distributed ray tracing of highly complex models. *Proc. Eurographics Workshop on Rendering*, 277– 288.

[WDG02] WALTER, B., DRETTAKIS, G., GREENBERG, D.P. 2002. Enhancing and optimizing the render cache. *Proc. Eurographics Workshop on Rendering*, 37–42.

[WDP99] WALTER, B., DRETTAKIS, G., PARKER S. 1999. Interactive rendering using render cache. *Proc. Eurographics Workshop on Rendering*, 19–30.

[WS99] WARD, G., SIMMONS, M. 1999. The Holodeck Ray Cache: An Interactive Rendering System for Global Illumination in Nondiffuse Environments, *ACM Trans. Graph.* 18, 4, 361-398.

[WLWD03] WOOLLEY, C., LUEBKE, D., WATSON, B.A., DAYAL, A. 2003. Interruptible rendering. *Proc. ACM Interactive 3D Graphics*, 143–151.