Image-Based Bidirectional Scene Reprojection

Kenneth Tse[1] Lei Yang[1] Pedro V. Sander[1] Jason Lawrence[2] Diego Nehab[3,4] Hugues Hoppe[3]

Hong Kong UST[1] University of Virginia[2] Microsoft Research[3] IMPA[4]

Document version: May 12, 2010
Technical Report #HKUST-CS10-06

Abstract

We introduce a general purpose method for increasing the framerate of real-time rendering applications. Whereas many existing temporal upsampling strategies only reuse information from previous frames, our bidirectional technique reconstructs intermediate frames from a pair of consecutive rendered frames. This significantly improves the accuracy of interpolated frames since very few pixels are mutually occluded in both frames, but does introduce a small amount of lag in the resulting image sequence (typically 1–2 frames). We present two versions of this basic algorithm. The first is appropriate for fill-bound scenes as it limits the number of expensive shading calculations, but requires processing the scene geometry at each intermediate frame. The second version lowers both shading and geometry computations by warping the pair of rendered images according to the scene depth and an estimate of the scene flow obtained through a simple iterative search. We demonstrate that our method offers substantial performance improvements (3–4$\times$) for a variety of applications, including vertex-bound and fill-bound scenes, multi-pass effects, and motion blur.

1 Introduction

Driven by the programmability and performance of dedicated graphics hardware, modern real-time rendering systems incorporate more computationally intensive pixel shading and larger geometric models than ever before. Additionally, mobile graphics platforms have emerged that offer exciting application opportunities, but possess far less computational power than desktop computers.

Providing high-quality rendered content on any of these platforms or porting applications from one platform to another requires general purpose methods for trading performance (framerate) for accuracy. A popular trend is to exploit the natural temporal and spatial coherence in animated image sequences by reusing the results of expensive shading calculations at nearby frames and pixels. The reverse reprojection scheme proposed by Nehab et al. [2007] allows *pulling* shading information from the previously rendered frame to accelerate the computation of the current frame. This approach provides an adjustable trade-off between performance and image quality and has proved useful in practice, including in the *Gears of War II* video game to accelerate low-frequency lighting effects.

However, a fundamental limitation of existing real-time reprojection techniques [Nehab et al. 2007; Scherzer et al. 2007; Sitthi-amorn et al. 2008a] is that they incur a drop in performance whenever there are disoccluded regions in the scene — elements visible in the current frame that were not visible in the preceding frame. Furthermore, the number of disoccluded pixels typically varies over time which can lead to undesirable fluctuations in framerate. Additionally, these multi-pass techniques are designed to reduce the average number of pixel shader invocations and are not suitable for accelerating vertex-bound scenes since they offer no reduction in geometry processing or rasterization.

We propose a *bidirectional* reprojection method for temporally upsampling rendered content that combines information from both the previous and next frames to achieve a higher and more stable framerate. Inspired by video compression algorithms, our approach is to insert interpolated frames between pairs of rendered frames. We will also adopt the terminology used in the video compression literature and will refer to rendered frames as *intra-* or simply *I-frames* and interpolated frames as *bidirectional predicted-* or *B-frames*. We observe that in the great majority of cases, the portion of the scene visible at each pixel in a B-frame is either visible in one of the I-frames or both, allowing reliable high-quality interpolation.

Our approach gives an adjustable trade-off between accuracy and performance without requiring manual profiling and optimization of shader code or scene geometry. This makes it particularly well suited for porting real-time applications to different architectures and display formats. The downside to our approach is that it introduces a lag in the resulting image sequence. However, we present a careful analysis of this lag and show that it is small, typically only one or two I-frames, and therefore the method is suitable for many real-time rendering applications.

We present two algorithms that fall within this basic bidirectional reprojection framework. Our *scene-assisted* algorithm is a straightforward extension of existing reverse reprojection methods and requires rasterizing the scene geometry in order to reconstruct each B-frame. This technique provides a performance improvement for fill-bound scenes in which evaluating the pixel shading comprises a significant portion of the rendering budget. As compared to conventional single-direction reprojection, it achieves superior image quality and higher framerates since combining information from multiple frames drastically reduces the disocclusion rate. Moreover, even in the rare situations where a pixel is not visible in either I-frame it allows forming an acceptable approximation that avoids ever having to evaluate the pixel shader.

We also describe an *image-based* algorithm which is a larger departure from prior work and allows accelerating both vertex- and fill-bound scenes. It reconstructs B-frames by warping the I-frames according to their corresponding depth maps and 3D scene flow, which is estimated using a simple iterative search performed in a fragment shader. The geometry is rasterized only at the I-frames and the complexity of computing the B-frames is proportional to the number of pixels in the framebuffer. Furthermore, we show how to achieve a stable framerate by interleaving the I-frame computation with the much less expensive reconstruction and display of B-frames. In many cases, our image-based algorithm produces B-frames that are nearly indistinguishable from reference images while providing a 3- to 4-fold increase in framerate.

2 Previous work

Data reuse Exploiting spatio-temporal coherence in order to render animated image sequences more efficiently has been extensively studied in both offline and interactive rendering systems [Cook et al. 1987; Badt 1988; Chen and Williams 1993; Bishop et al. 1994; Adelson and Hodges 1995; Mark et al. 1997; Walter et al. 1999; Bala et al. 1999; Ward and Simmons 1999; Havran et al. 2003; Tawara et al. 2004], as well as hybrid CPU-GPU ray-based systems [Simmons and Séquin 2000; Stamminger et al. 2000; Walter et al. 2002; Woolley et al. 2003; Gautron et al.

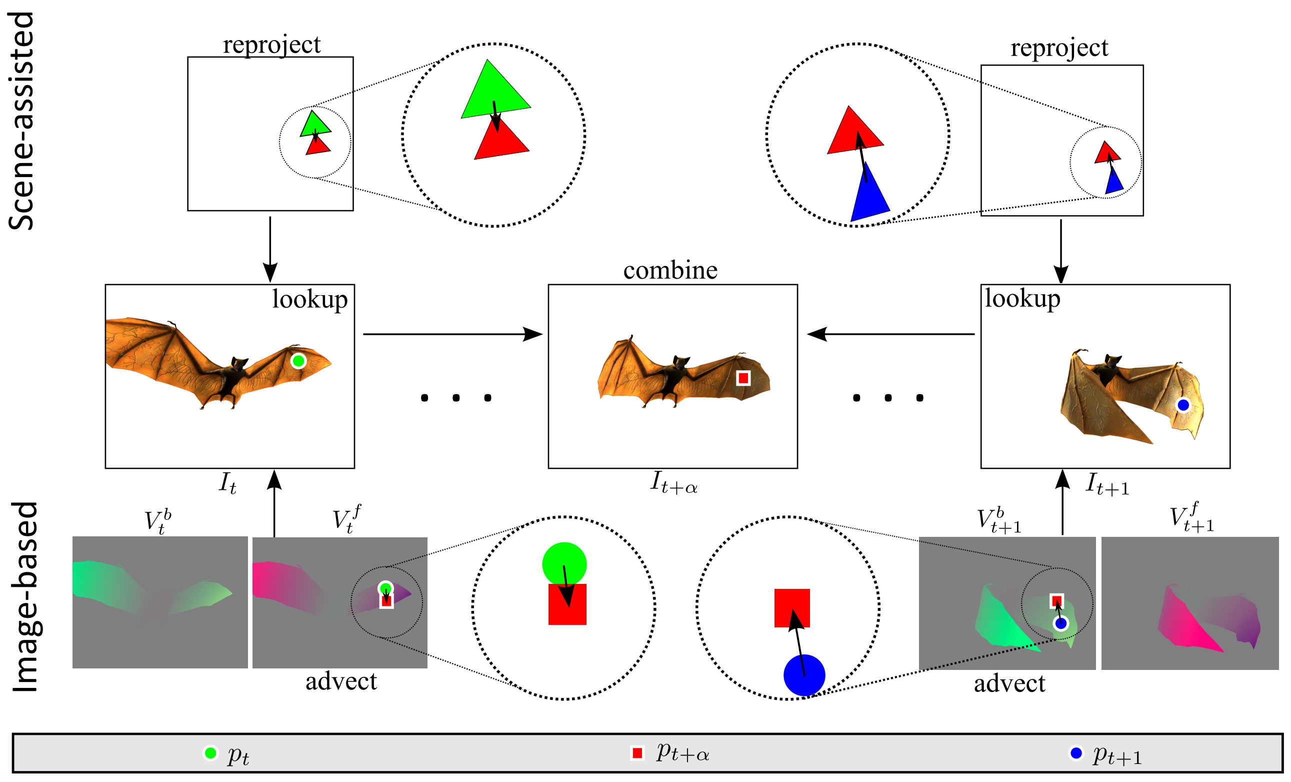


Figure 1: *Overview of scene-assisted and image-based algorithms for reconstructing one pixel in a B-frame:* $I_{t+\alpha}[p_{t+\alpha}]$.

2005; Zhu et al. 2005; Dayal et al. 2005]. For scanline-based rendering systems, recently proposed reverse reprojection methods allow reusing the shading from the previous frame to reduce the cost of computing the shading in the current frame [Nehab et al. 2007; Scherzer et al. 2007; Sitthi-amorn et al. 2008a,b]. Another approach exploits temporal coherence to amortize the computation of supersampled frames [Yang et al. 2009]. Finally, there are hybrid methods that exploit not only temporal, but also spatial coherence [Herzog et al. 2010].

Image warping and interpolation Image warping algorithms play a key role in image morphing and image-based rendering systems [Beier and Neely 1992; Chen and Williams 1993; McMillan and Bishop 1995; Seitz and Dyer 1996; Vedula et al. 2002; Fitzgibbon et al. 2005; Stich et al. 2008a,b]. These techniques often use sparse feature correspondences obtained either from user input or automatic feature extraction algorithms. A notable exception is the Moving Gradients system proposed by Mahajan et al. [2009] which computes space-time paths through an image sequence and uses gradient domain techniques to reconstruct the pixel values at intermediate frames. In contrast to our work, Moving Gradients targets a harder problem in which accurate and dense scene flow and scene depth is not available.

Mark et al. [1997] describe a system that upsamples rendered frames by warping images with the aid of scene depth. However, this method only considers changes in viewpoint. A related method is due to Didyk et al. [2010] which targets high-refresh-rate displays. Similar to our approach, exact scene flow between neighboring I-frames is computed using the available 3D scene geometry. This flow field is used to compute an image warp defined over a coarse grid superimposed over the framebuffer. Occlusion artifacts are diminished by applying an adaptive spatial blur. By reconstructing a dense scene flow field that relates each B-frame to its adjacent I-frames, our technique produces higher quality interpolated content and is thus more suitable for applications targeting lower framerates where B-frames are more easily visible. We present a direct comparison to this method in Section 7.

Video compression Another related set of methods are video compression algorithms. Standards such as H.264, AVC, MPEG-4 [Wiegand et al. 2003; Sullivan and Wiegand 2005] all incorporate some form of motion compensation, in which the motion of small windows of pixels (blocks) between consecutive frames is estimated and used to further reduce the bitrate. These techniques also encode video frames using information from multiple reference I-frames in a way analogous to bidirectional reprojection.

3 Overview

Our basic approach is to render the full 3D scene at I-frames using conventional methods and then insert interpolated B-frames between these to achieve a higher framerate. As compared to standard single-direction reprojection methods, this approach significantly lowers disocclusion artifacts by virtue of using information from two viewpoints instead of one. The interpolation process is guided by *scene flow*, the 3D velocities of visible surface points between two frames, which indicates where and how to pull shading information from the I-frames to reconstruct each B-frame.

We will use F_t to denote the framebuffer of the I-frame rendered at time $t \in \mathbb{Z}$. Between successive I-frames F_t, F_{t+1}, we compute $n-1$ B-frames, corresponding to times $t+\alpha$ with $\alpha \in \frac{1}{n}, \ldots \frac{n-1}{n}$. Because this series of B-frames will be displayed before the I-frame at time $t+1$, our upsampling algorithm introduces a lag of roughly one I-frame with respect to a conventional rendering system. Section 6 provides a detailed analysis of this lag.

We let the symbol $p = (p_x, p_y)$ denote the 2D coordinate of a pixel in clip space. The third coordinate (depth) is available during geometry rasterization and in the depth buffer Z. We use $\bar{p} = \big(p_x, p_y, Z[p]\big)$ to denote its corresponding 3D coordinate. Because models may deform and the viewpoint may change over time, we must account for changes in clip spaces. We let $\pi_{t \to t'}$ denote

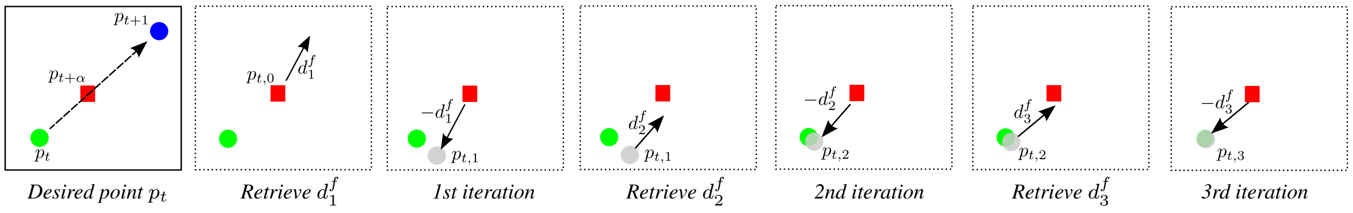


Figure 2: *Three iterations of the search algorithm used to estimate the scene flow at one pixel in a B-frame with respect to an I-frame in the forward direction.*

the transformation that maps the surface point $\bar{p}_t$ at time t into the clip space at time t'. This transformation is easily computed during rasterization of the frame at time t, by supplying the animation and camera parameters for time t', and leveraging hardware interpolation of per-vertex attributes [Nehab et al. 2007].

We introduce two interpolation algorithms:

- *Scene-assisted interpolation* computes the exact scene flow between each B-frame and its pair of neighboring I-frames by rasterizing the scene geometry (without shading) at each B-frame. (Section 4)

- *Image-based interpolation* only computes the exact scene flow between adjacent I-frames and uses a simple iterative search to estimate the scene flow between each B-frame and its neighboring I-frames. This sacrifices accuracy in the interpolated frames, but avoids rasterizing the geometry more than once per I-frame. (Section 5)

In the following sections we describe these two interpolating strategies in detail. Section 7 presents results along with a comparison of the two approaches.

4 Scene-assisted interpolation

This technique renders the depth of the scene at each B-frame (no shading) and uses reprojection to reconstruct the shading from the two adjacent I-frames (Figure 1, top). Because the shading is only ever computed at the I-frames, this approach can increase the framerate of fill-bound scenes.

I-frame Each I-frame buffer $F_t = (I_t, Z_t)$ consists of an $RGBA$ color image I_t and a depth buffer Z_t, obtained using straightforward rasterization. Before reconstructing the B-frames within the interval $[t, t+1]$, we must first rasterize F_{t+1}. Note that F_t will have already been generated during rendering the interval $[t-1, t]$.

B-frame To reconstruct the B-frame image $I_{t+\alpha}$, we rasterize the scene geometry at time $t+\alpha$ and perform reprojection into both adjacent I-frames (F_t, F_{t+1}) (see Figure 1). The position of the 3D surface point visible at each pixel $\bar{p}_{t+\alpha}$ with respect to the camera at time t and $t+1$ is computed in the vertex shader and interpolated during rasterization [Nehab et al. 2007]. Note that the necessary camera parameters and animation parameters at time $t+1$ are known since frame F_{t+1} has already been rendered. The reprojection step compares the depth of $\pi_{t+\alpha\to t}(\bar{p}_{t+\alpha})$ and $\pi_{t+\alpha\to t+1}(\bar{p}_{t+\alpha})$ to the depth stored at $Z_t[\pi_{t+\alpha\to t}(\bar{p}_{t+\alpha})]$ and $Z_{t+1}[\pi_{t+\alpha\to t+1}(\bar{p}_{t+\alpha})]$, respectively, in order to identify possible occlusions [Nehab et al. 2007]. If the surface point is visible in only one I-frame, we simply set $I_{t+\alpha}(p_{t+\alpha})$ to its shaded color there. If it is visible in both, we blend the shaded colors based on α. If it is visible in neither, we follow one of two approaches:

- *Shade-on-miss* simply evaluates the pixel shader to obtain an accurate result.

- *Closest-on-miss* uses the color from the nearest buffer. In other words, the depth values at the reprojected positions in Z_t and Z_{t+1} are compared and the shading associated with whichever is smaller is used.

In our results, we used the closest-on-miss approach. This strategy maps well to the SIMD architecture of modern graphics hardware since it avoids any conditional execution and branch divergence at neighboring pixels. Furthermore, we have found that it produces results that are nearly indistinguishable from the more conservative shade-on-miss approach largely due to the rarity of this “double miss” case.

5 Image-based interpolation

Our image-based interpolation algorithm also reconstructs B-frames at uniformly spaced time locations in the interval $[t, t+1]$, but does so without rasterizing the scene geometry. Instead, we use the 3D scene flow between adjacent I-frames to drive the interpolation process. Specifically, as F_t is rendered we compute and store a *forward flow field* V_t^f which encodes the motion of the scene at each pixel between I-frames at times t and $t+1$ (the 3D motion relative to the image plane at time t)

$$V_t^f[p] = \pi_{t\to t+1}(\bar{p}_t) - \bar{p}_t \ . \tag{1}$$

Similarly, we compute a *backward flow field* V_{t+1}^b during rendering F_{t+1} which encodes the per-pixel relative scene motion between I-frames at times $t+1$ and t:

$$V_{t+1}^b[p] = \pi_{t+1\to t}(\bar{p}_{t+1}) - \bar{p}_{t+1} \ . \tag{2}$$

These flow fields are computed using the same reprojection technique described above and originally proposed by Nehab et al. [2007].

I-frame At time t, the algorithm first renders V_t^f. It then renders the buffers I_{t+1}, Z_{t+1}, V_{t+1}^b associated with time $t+1$ in a single rendering pass using multiple render targets. Note that the buffers I_t and Z_t, which will also be needed to reconstruct the B-frames in the interval $[t, t+1]$, are available after rendering the I-frame at time $t-1$.

B-frame For each pixel in each B-frame we wish to find the pixel coordinates of the same surface point (if visible) in the adjacent I-frames so that we can reuse those colors. In what follows, we describe an iterative search for estimating these positions that can be efficiently implemented on graphics hardware and discuss some useful optimizations.

5.1 Iterative search

For each pixel $p_{t+\alpha}$ in B-frame $F_{t+\alpha}$, our goal is to compute the corresponding locations $p_t = \pi_{t+\alpha\to t}(\bar{p}_{t+\alpha})$ and $p_{t+1} = \pi_{t+\alpha\to t+1}(\bar{p}_{t+\alpha})$ in the neighboring I-frames F_t and F_{t+1}, respectively. Since we do not have access to the scene geometry and camera parameters at time $t+\alpha$ and thus cannot perform exact reprojection as in the scene-assisted case, we will instead approximate these positions using a greedy search directed by the forward and backward flow fields.

Forward direction We will assume that the pixel coordinate $p_{t+\alpha}$ and its corresponding coordinate p_t in I-frame B_t are related to one another according to the fractional displacement along the forward flow field $\alpha V_t^f[p_t]$ (leftmost image in Figure 2). Specifically,

$$p_{t+\alpha} = p_t + \alpha V_t^f[p_t].xy \; . \tag{3}$$

This is equivalent to assuming that all surface points undergo linear motion relative to the *moving coordinate frame* associated with the clip space coordinate system at time $t+\alpha$. Under these conditions, if the 3D surface point at pixel $p_{t+\alpha}$ is visible in I-frame B_t, then at least one solution to Equation 3 must exist (up to sampling error). Note that multiple solutions may exist, since other points visible in B_t may also map to $p_{t+\alpha}$ but are occluded in this frame.

As illustrated in Figure 2, we estimate p_t using a simple search. We start with

$$p_{t,0} = p_{t+\alpha} \; , \tag{4}$$

and iteratively compute

$$p_{t,i} = p_{t+\alpha} - d_i^f \quad \text{where} \quad d_i^f = \alpha V_t^f[p_{t,i-1}].xy \; . \tag{5}$$

So that this algorithm will map well to a SIMD architecture, we always terminate the search after a fixed number of m iterations. We compute the clip space depth of the computed surface point as

$$z^f = Z_t[p_{t,m}] + \alpha V_t^f[p_{t,m}].z \; . \tag{6}$$

Finally, a measure of the screen space error is given by

$$e^f = \left\| \left(p_{t,m} + \alpha V_t^f[p_{t,m}].xy\right) - p_{t+\alpha} \right\| \; . \tag{7}$$

Backward direction In parallel, we perform the same process as above, but with respect to the I-frame at time $t+1$. Specifically,

$$\begin{aligned} p_{t+1,0} &= p_{t+a} \; , & (8) \\ p_{t+1,i} &= p_{t+\alpha} - d_i^b \; , & (9) \\ d_i^b &= (1-\alpha)\; V_{t+1}^b[p_{t+1,i-1}].xy \; . & (10) \end{aligned}$$

Similarly, we compute the depth with respect to the clip space coordinate system at frame $t+\alpha$ as

$$z^b = Z_t[p_{t+1,m}] + (1-\alpha)\; V_{t+1}^b[p_{t+1,m}].z \; , \tag{11}$$

and screen space error

$$e^b = \left\| \left(p_{t+1,m} + (1-\alpha) V_{t+1}^b[p_{t+1,m}].xy\right) - p_{t+\alpha} \right\| \; . \tag{12}$$

5.2 Visibility and shading

After these searches terminate, we test whether the resulting screen space errors are within a threshold ($e^f < \epsilon_1$ and $e^b < \epsilon_1$).

(1) If they are both below this threshold and have similar depths $\left(|z^f - z^b| < \epsilon_2\right)$, we conclude that they refer to the same 3D surface point and a straightforward approach would be to simply blend the two colors according to α:

$$(1-\alpha) I_t[p_{t,m}] + \alpha I_{t+1}[p_{t+1,m}] \; . \tag{13}$$

However, we have found that this introduces undesirable blurring since the two surface points will never be identical. Instead, we identify the point with the smallest screen-space error and project that point into the other I-frame before blending the colors. (Due to the exact flow fields, we can perform this mapping precisely.) Thus, in the case that $e^f < e^b$, we compute the blended color as:

$$(1-\alpha) I_t[p_{t,m}] + \alpha I_{t+1}[p_{t,m} + V_t^f[p_{t,m}].xy] \; . \tag{14}$$

Conversely, if $e^b \leq e^f$, we compute the blended color as:

$$(1-\alpha)\; I_t\left[p_{t+1,m} + V_{t+1}^b[p_{t+1,m}].xy\right] + \alpha\; I_{t+1}[p_{t+1,m}] \; . \tag{15}$$

Note that we must ensure the point is visible in the other I-frame. In the rare event that it is not, we fall back to only using the results from the frame with the least screen-space error.

(2) If both errors are below ϵ_1 but have different depths, we select the color closest to the camera (since it occludes the other point). We still map that same point into the other I-frame, and if it is visible there as well, we blend the two colors as in step (1) above; this is essentially a "second chance" to find the appropriate point in the other frame.

(3) In the rare case where both errors exceed our tolerance, we simply apply the blending procedure in step (1) to the solutions of both searches.

5.3 Additional search initializations

The bidirectional search described above generally produces good matches. However, there are situations where the starting points of both searches lead to problems. We address this by performing the search from additional starting points and using whichever solution has the smallest screen space error in the end. These additional starting conditions are described next.

Dual initialization One difficult situation arises along the silhouettes of an object that is both rotating and translating. In the previous I-frame, the surface is visible at the pixel but if one subtracts the motion vector one falls off the object, and in the next I-frame the surface is no longer under the pixel. This can be seen near the silhouette of the globe in Figure 4.

To address this situation, we introduce one additional starting point for each of the two iterative searches. The basic idea is to initialize the search in one I-frame by using the velocity vector retrieved from the other I-frame:

$$p'_{t,0} = p_{t+\alpha} + \alpha V_{t+1}^b(p_{t+\alpha}) \; , \tag{16}$$

$$p'_{t+1,0} = p_{t+\alpha} + (1-\alpha) V_t^f(p_{t+\alpha}) \; . \tag{17}$$

We follow the same iterative procedure described previously from this starting position.

Latest-frame initialization The aforementioned strategies only consider the motion vectors at the current pixel. However, for all but the first B-frame in any interval, we can exploit the scene's temporal coherence and initialize the search using the result d_i from the previous B-frame. Note, however, that the offset d_i must be scaled according to α in order to account for the elapsed time between neighboring B-frames.

In this case, the initialization of the forward search becomes:

$$p_{t,0} = p_{t+\alpha} - d_0^f \; , \quad \text{where} \quad d_0^f = \frac{\alpha}{\alpha'} d_i'^f \; , \tag{18}$$

where $d_i'^f$ is the offset computed in the previous B-frame at time $t+\alpha'$. The initialization of the backward search is similar:

$$p_{t+1,0} = p_{t+\alpha} - d_0^b \; , \text{ where } d_0^b = \frac{1-\alpha}{1-\alpha'} d_i'^b \; . \tag{19}$$

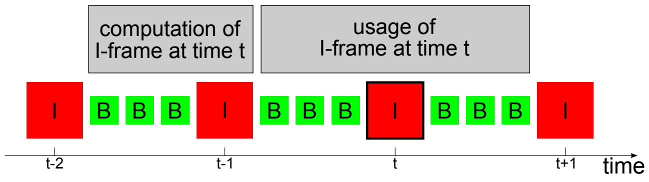


Figure 3: *The amortized computation of the I-frame at time t takes place within the $[t-2, t-1]$ interval.*

where $d_i'^b$ is the offset computed in the previous B-frame at time $t+\alpha'$. We observed the best results if we consider all of the offsets within a small fixed neighborhood around the pixel and use whichever has the smallest depth. In particular, this improves the robustness of the search near depth discontinuities. For example, consider the north pole of the globe in Figure 4. Using motion vectors from nearby pixels allows the algorithm to "find" the globe, and consequently properly initialize the search using a motion vector that is consistent with the underlying motion of this object.

6 Partitioned rendering and lag

The relatively expensive computation of the I-frames (as compared to the cost of computing B-frames) results in an uneven rendering load. To address this problem, we amortize the computation associated with the I-frame at time t along the B-frame rendering interval $[t-2, t-1]$. Note that the rendering associated with the I-frame at time t must be fully completed by time $t-1$, since it is required to render the B-frames in the interval $[t-1, t]$ (see Figure 3).

The amortization is achieved by simply partitioning the rendering tasks associated with the scene as evenly as possible and interleaving these rendering tasks with the rendering of B-frames in that interval. This process is application dependent. For scenes that have a large number of shaded objects with roughly uniform shading costs, amortization can be achieved by simply uniformly partitioning the objects across the B-frames in the $[t-2, t-1]$ interval, shading them according to rendering parameters of the I-frame, and accumulating the results on the appropriate render targets. It is important to note that most of the rendering cost of the B-frames comes from the amortized shading of the I-frames.

Lag In this amortized scenario, the lag is two I-frames, as illustrated in Figure 3. In practice, this lag is acceptable for interactive applications such as mesh visualization, procedural animation, and most computer games. If *immediate* feedback is desired (e.g., a shooting scene in a first-person shooter game), any additional lag could be problematic, and therefore this system may not be applicable. However, it is worth noting that, in a traditional rendering applications, there is typically a lag of at least one I-frame: the time between when the scene information is gathered for rendering until the time when the frame is ready for display. With v-sync enabled, this lag is increased by up to one additional frame since the system has to wait for the refresh interval before swapping buffers. Finally, real-time rendering systems often queue frames inside the driver and on the GPU for added efficiency (e.g., Direct3D buffers 3 frames by default).

7 Results

In this section we discuss different usage scenarios for bidirectional reprojection along with the quality/speed trade-offs associated with using the proposed techniques. All results were generated on an Intel Core Duo 3GHz CPU with 2GB of RAM and an NVIDIA GeForce 8800 GTX graphics card.

The simple *globe* scene in Figure 4 demonstrates that both the scene-assisted and image-based methods can properly handle an animation that involves translation, rotation, and scaling. As the model animates, our algorithms for rendering the three intermediate B-frames $I_{t+0.25}$, $I_{t+0.5}$, and $I_{t+0.75}$ correctly pull information

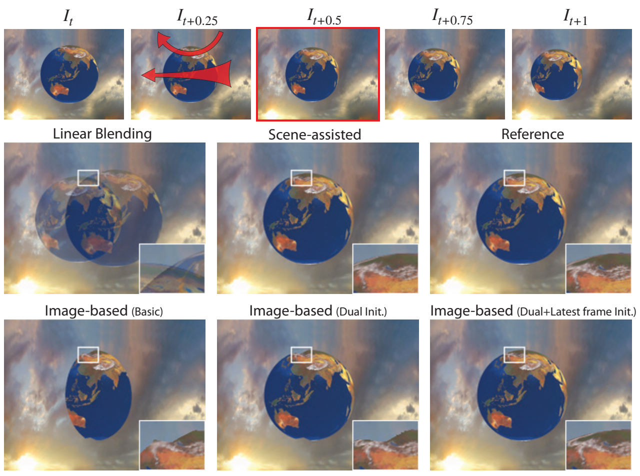


Figure 4: *Bidirectional reprojection on an animation that involves translation, rotation, and scaling.*

from I_t, and I_{t+1}. In this simple animation, all surface points rendered in the B-frames were visible in at least one of the I-frames, thereby allowing the scene-assisted algorithm to always locate the correct pixel in the I-frame(s). The basic image-based algorithm was unable to find the appropriate surface points for some of the pixels close to the globe's silhouette. However, after employing the additional search initializations described in Section 5.3, the search succeeds.

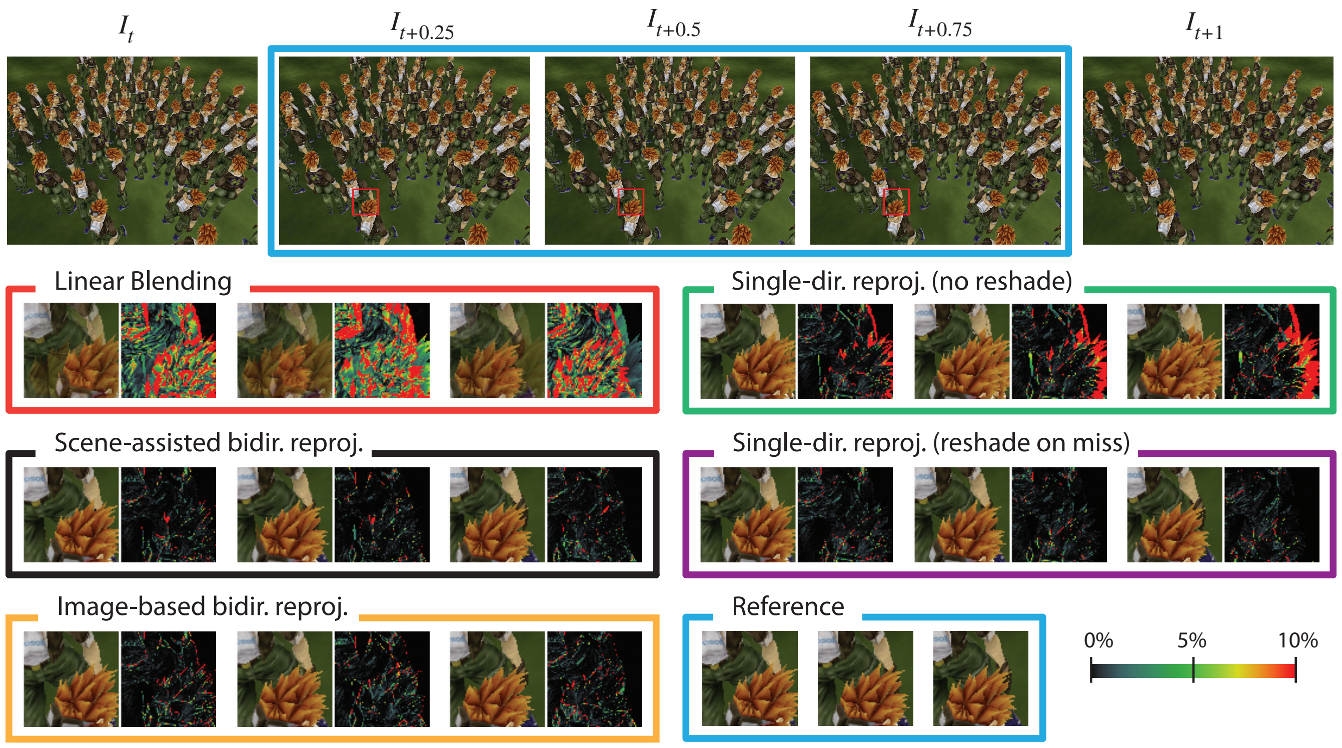
7.1 Applications

Figures 5–8 present results for both scene-assisted and image-based bidirectional reprojection for several different scenes. We have found that using three iterations of the image-based algorithm suffices for convergence on all scenes we tested. For each scene, we compared our approach with two variants of traditional reprojection: one that reshades disoccluded regions (i.e., it reshades *cache misses*) and one that doesn't. Traditional reprojection with reshading can either be performed in a single pass or in multiple passes that take advantage of early-Z culling [Sitthi-amorn et al. 2008a]. We always used whichever method was fastest for the given application (single-pass for the vertex-bound terrain scene and multi-pass for the walking scene). We also show comparisons with naive linear blending, which causes clear ghosting artifacts. For each result, we show graphs measuring rendering time and quality (in MSE and SSIM) compared to the reference images over a given animation sequence. We also show close-ups and difference images for three B-frames $I_{t+0.25}$, $I_{t+0.5}$, and $I_{t+0.75}$. Note that one out of every four frames is an I-frame and therefore has no error. Hence the periodic pattern visible in the quality measurement graphs.

In some applications, partitioned rendering achieved a more stable framerate than others (see scenario descriptions below). Nevertheless, even in the scenes where a perfectly uniform amortization is not achievable, the rendering time of each B-frame using our approach is substantially faster than reshading the scene anew.

Next we describe each result in greater detail. Please refer to the accompanying video for animated renderings of these results.

Fill-bound scenes The *walking* scene of Figure 5 is an example of a fill-bound scene consisting of moving characters over a floor that is shaded by an expensive procedural noise function. Fill-bound scenes are common in real-time applications (e.g., computer games), and bidirectional reprojection can significantly reduce their rendering cost by reducing the number of expensive pixels that need to be processed. The bottom row of Figure 5 shows quality and performance results of this technique for a representative animation segment. Note that bidirectional reprojection is nearly 3× faster



I_t
$I_{t+0.25}$
$I_{t+0.5}$
$I_{t+0.75}$
I_{t+1}
Linear Blending
Single-dir. reproj. (no reshade)
Scene-assisted bidir. reproj.
Single-dir. reproj. (reshade on miss)
Image-based bidir. reproj.
Reference
0%
5%
10%

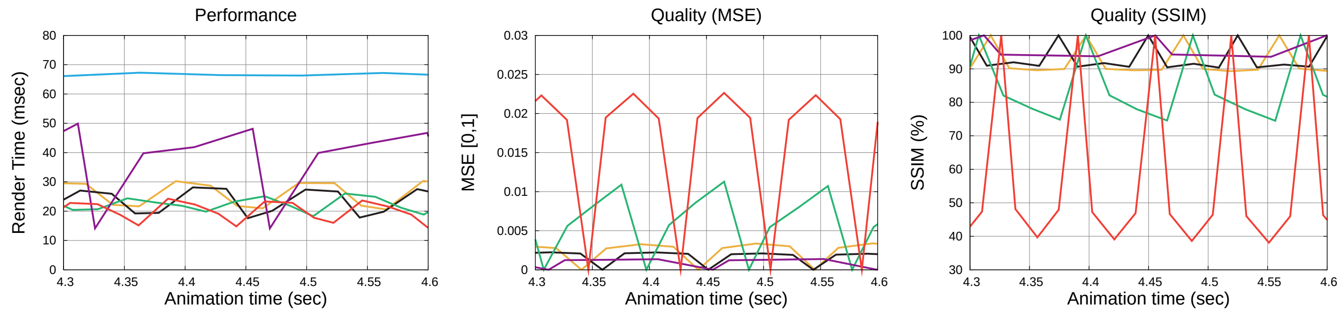


Figure 5: *Results of our algorithm on the walking scene. The lines in the plots are colored according to the color of the frame around the insets of the corresponding method.*

than shading the scene anew while producing high quality results. Even with severe disocclusion, it is clear from the inset difference images that our image-based technique is able to properly reproject the samples without any aid from the scene geometry. The results are only slightly inferior to our scene-assisted technique. Since our methods do not reshade any pixel in the intermediate frames, they are nearly as fast as the low-quality naive linear blending. Most of the cost in these frames come from the amortized shading of the I-frames (the characters were uniformly partitioned for amortized rendering). Using traditional single-direction reprojection results in cache misses for every disoccluded region from the previous I-frame. Therefore, it produces significantly worse results unless the pixels are shaded anew, in which case performance deteriorates significantly.

Vertex-bound scenes For large meshes, such as the 1M-triangle *terrain* scene of Figure 6, most of the rendering budget is consumed during vertex processing. For these types of scenes, we can also provide a framerate improvement when using the image-based interpolation approach, which achieves nearly a $3\times$ speedup for the terrain. Note that the errors are higher in this scene due to the use of a high-frequency "noisy" pixel shader. In practice, however, these differences are indistinguishable in the real-time animation (see accompanying video), and such shaders present no problems with our bidirectional reprojection framework. With our technique, all vertex processing is only performed when rendering the I-frames rather than for all frames in the animation. For this example, the terrain is partitioned into a square grid of cells for amortized rendering. Since the bottleneck is on vertex processing, our scene-assisted approach and traditional single-direction reprojection with reshading on cache misses are slow and therefore not applicable. It is interesting to note that, when using an inexpensive pixel shader as in this example, the scene can be shaded anew at each B-frame by simply computing and reprojecting the pixel shader *inputs* (e.g., surface normal and texture coordinates) rather than the final color. The B-frames can then shade these pixels with dynamic lighting conditions without incurring the expensive additional vertex processing.

Multi-pass rendering effects Many computer graphics rendering effects require multiple rendering passes to intermediate temporary textures prior to generating the final rendered result. The NVIDIA *human head* demo in Figure 7 is such an example. It uses a sum-of-Guassians formulation of subsurface scattering which is

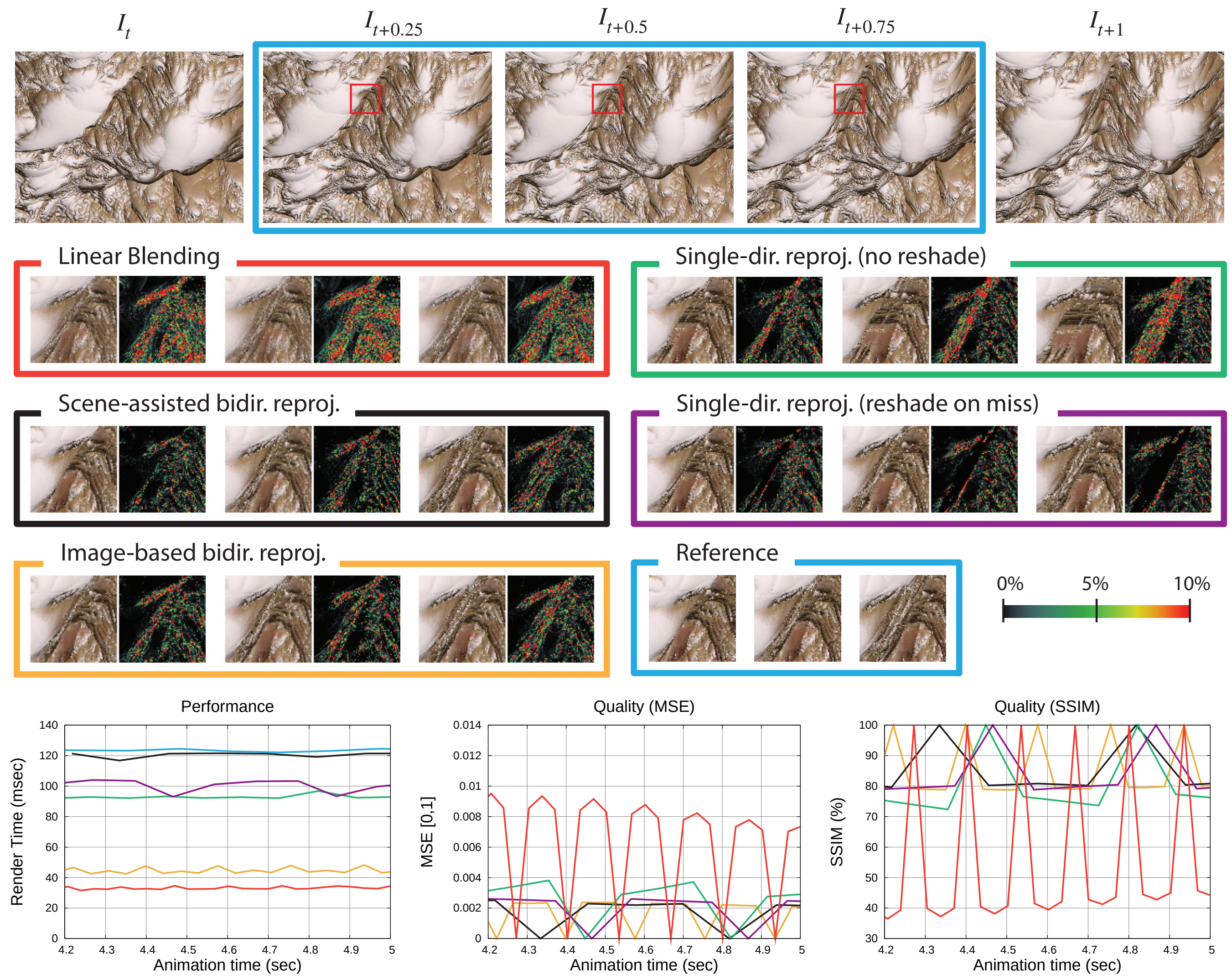


Figure 6: *Results of our algorithm on the terrain scene. The lines in the plots are colored according to the color of the frame around the insets of the corresponding method.*

computed in texture-space using a series of render-to-texture passes (Refer to d'Eon and Luebke [2007] for details.) The key advantage of bidirectional reprojection in such multi-pass scenes is that since it does not reshade on a cache miss, none of the intermediate passes need to be rendered at each B-frame. They are only needed when shading the I-frames. Therefore traditional single-direction reprojection with reshading is not suitable, whereas single-direction reprojection without reshading is unable to properly handle disoccluded regions. For this example, the more accurate motion flow from the scene-assisted approach yields results that are a bit better than the image-based method. The rendering costs are similar since geometry processing is not a bottleneck in this application.

Motion blur Our technique can also be used to render scenes with motion blur. B-frames immediately before and after a given I-frame are composited together to generate a motion blurred scene with little added cost. We use the *walking* scene again for this example, although using a different animation segment. Accumulating ten B-frames per I-frame (Figure 8), we achieve a $5\times$ speedup relative to brute-force, with negligible quality loss. The choice between using scene-assisted and image-based interpolation represents a (smaller) trade-off between speed and quality in this case, as illustrated by these graphs. Single-direction reprojection with reshading achieves better results, but it is significantly slower since it has to reshade on cache misses. Single-direction reprojection without reshading is very fast, but again suffers from significant artifacts at disoccluded regions.

7.2 Comparison to frame upsampling

Figure 9 shows example B-frame renderings of our approach compared to that of Didyk et al. [2010]. We must stress that this is not a completely fair comparison, since the method of Didyk et al. [2010] is targeted at higher-frame-rate displays, where the individual B-frames are hardly perceived. We included this comparison to demonstrate that our image-based bidirectional reprojection technique is the first real-time technique that interpolates frames in image-space with high enough quality for rendering on standard displays with common refresh rates.

8 Conclusion

We have introduced a new real-time technique for temporally upsampling rendered image sequences. As compared to single-reprojection methods, our approach reduces artifacts due to disocclusion by using information from both the previous and the fol-

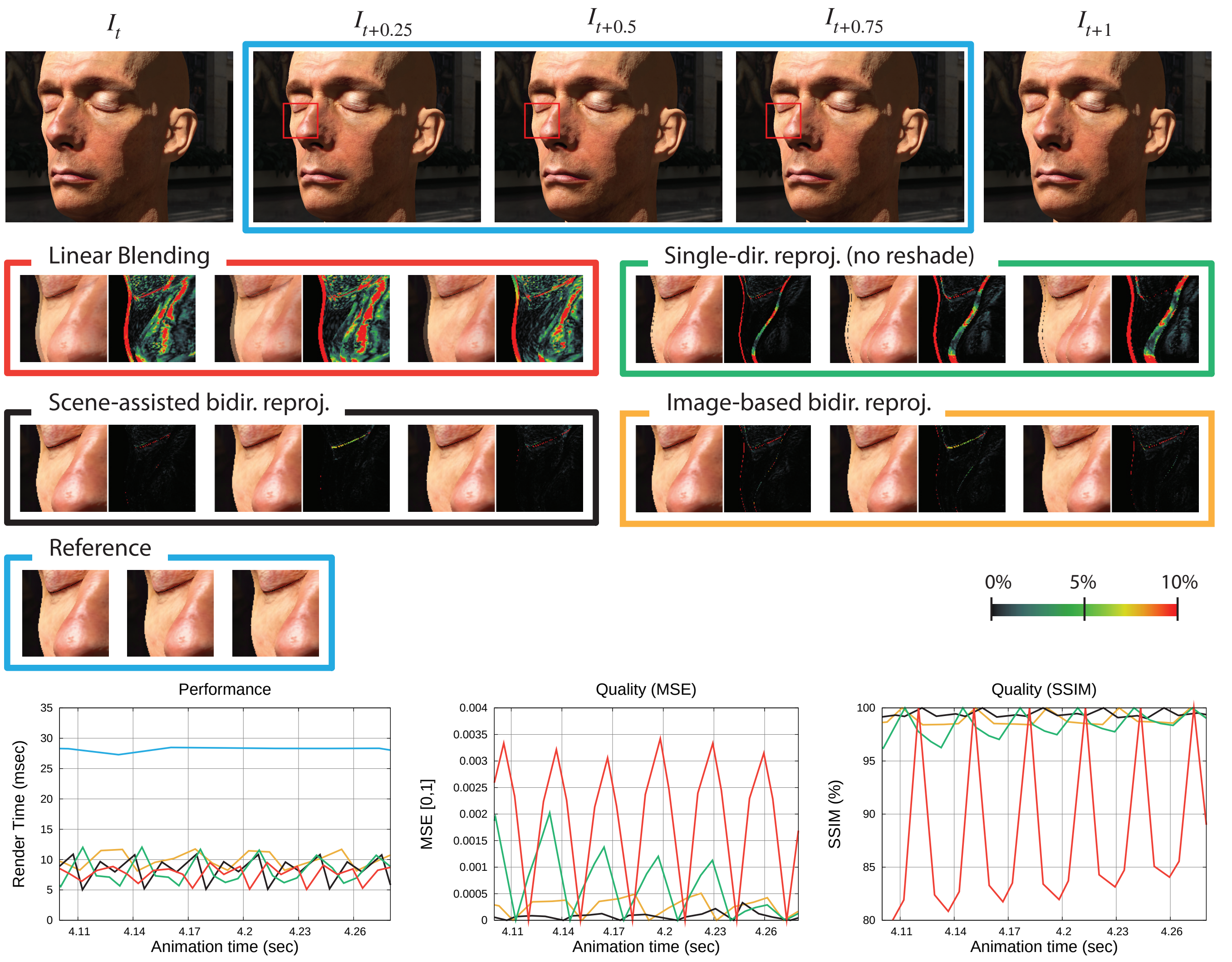


Figure 7: *Results of our algorithm on the NVIDIA human head scene. The lines in the plots are colored according to the color of the frame around the insets of the corresponding method.*

lowing rendered frames. We presented two algorithms to transfer the information from these I-frames to the interpolated frames. One computes the correspondence robustly using geometry reprojection. The other uses an image-space search. We also described how to amortize the computation of these I-frames over multiple rendered frames, and presented results of having successfully applied our method to significantly speed-up rendering of scenes that are fill-bound, vertex-bound, and contain motion blur.

For future work, we would like to consider automatically partitioning the scene for amortization using a measure of expected rendering cost for the scene partitions and load balancing across frames. We also would like to consider a multi-layer extension of the approach in order to handle semi-transparent geometry.

References

ADELSON, S. J. and HODGES, L. F. 1995. Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications*, 15(3):43–52.

BADT, JR., S. 1988. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–132.

BALA, K., DORSEY, J., and TELLER, S. 1999. Radiance interpolants for accelerated bounded-error ray tracing. *ACM Transactions on Graphics*, 18(3):213–256.

BEIER, T. and NEELY, S. 1992. Feature-based image metamorphosis. *SIGGRAPH Comput. Graph.*, 26(2):35–42.

BISHOP, G., FUCHS, H., MCMILLAN, L., and ZAGIER, E. J. S. 1994. Frameless rendering: Double buffering considered harmful. In *Proc. of ACM SIGGRAPH 94*, ACM Press/ACM SIGGRAPH, pages 175–176.

CHEN, S. E. and WILLIAMS, L. 1993. View interpolation for image synthesis. In *Proc. of ACM SIGGRAPH 93*, ACM Press/ACM SIGGRAPH, pages 279–288.

COOK, R. L., CARPENTER, L., and CATMULL, E. 1987. The REYES image rendering architecture. *Computer Graphics (Proc. of ACM SIGGRAPH 87)*, 21(4):95–102.

DAYAL, A., WOOLLEY, C., WATSON, B., and LUEBKE, D. 2005. Adaptive frameless rendering. In *Eurographics Symposium on Rendering*, Rendering Techniques, Springer-Verlag, pages 265–275.

D'EON, E. and LUEBKE, D. 2007. Advanced techniques for realistic real-time skin rendering. In *GPU Gems 3*, Addison-Wesley Professional.

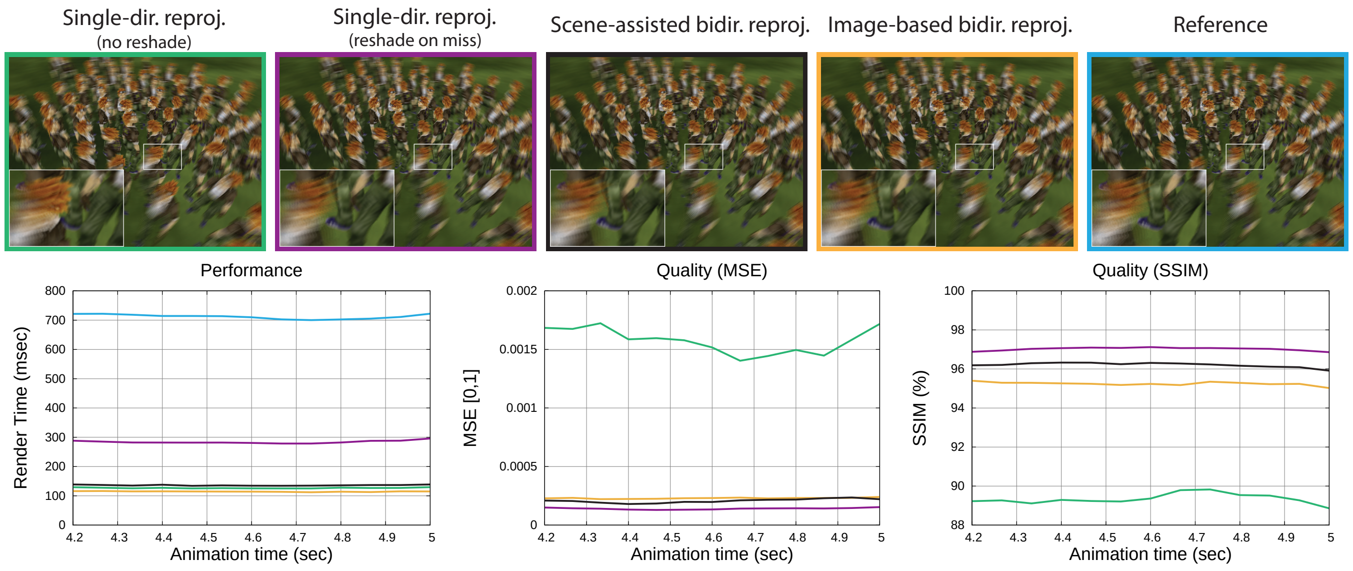


Figure 8: *Results of our algorithm on the walking scene with motion blur.*

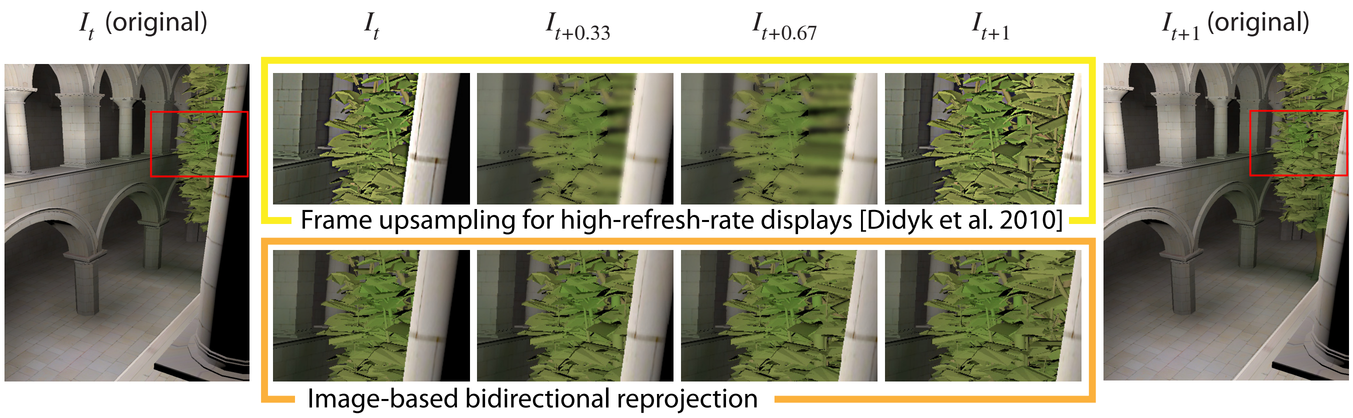

I_t (original)
I_t
$I_{t+0.33}$
$I_{t+0.67}$
I_{t+1}
I_{t+1} (original)
Frame upsampling for high-refresh-rate displays [Didyk et al. 2010]
Image-based bidirectional reprojection

Figure 9: *Comparison with the method of Didyk et al. [2010], which is suitable for high-frame-rate displays where individual interpolated frames are hardly visible. (Note that I-frames are not identical to ours because Didyk et al. [2010] use contrast enhanced images to counteract blurriness in the interpolated frames.)*

DIDYK, P., EISEMANN, E., RITSCHEL, T., MYSZKOWSKI, K., and SEIDEL, H.-P. 2010. Perceptually-motivated real-time temporal upsampling of 3d content for high-refresh-rate displays. *Comput. Graph. Forum (Proc. Eurographics 2010)*, 29(2).

FITZGIBBON, A., WEXLER, Y., and ZISSERMAN, A. 2005. Image-based rendering using image-based priors. *International Journal of Computer Vision*, 63(2):141–151.

GAUTRON, P., KŘIVÁNEK, J., BOUATOUCH, K., and PATTANAIK, S. 2005. Radiance cache splatting: A GPU-friendly global illumination algorithm. In *Eurographics Symposium on Rendering*, pages 55–64.

HAVRAN, V., DAMEZ, C., MYSZKOWSKI, K., and SEIDEL, H.-P. 2003. An efficient spatio-temporal architecture for animation rendering. In *Eurographics Symposium on Rendering*, Rendering Techniques, Springer-Verlag, pages 106–117.

HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., and SEIDEL, H.-P. 2010. Spatio-temporal upsampling on the GPU. In *Symposium on Interactive 3D Graphics and Games*, ACM.

MAHAJAN, D., HUANG, F.-C., MATUSIK, W., RAMAMOORTHI, R., and BELHUMEUR, P. 2009. Moving gradients: a path-based method for plausible image interpolation. *ACM Trans. Graph.*, 28(3):1–11.

MARK, W. R., MCMILLAN, L., and BISHOP, G. 1997. Post-rendering 3D warping. In *Symposium on Interactive 3D Graphics*, pages 7–16.

MCMILLAN, L. and BISHOP, G. 1995. Plenoptic modeling: an image-based rendering system. In *Proc. of ACM SIGGRAPH 95*, ACM, pages 39–46.

NEHAB, D., SANDER, P. V., LAWRENCE, J., TATARCHUK, N., and ISIDORO, J. R. 2007. Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware*, pages 25–35.

SCHERZER, D., JESCHKE, S., and WIMMER, M. 2007. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Eurographics Symposium on Rendering*, pages 45–50.

SEITZ, S. M. and DYER, C. R. 1996. View morphing. In *Proc. of ACM SIGGRAPH 96*, ACM, pages 21–30.

SIMMONS, M. and SÉQUIN, C. H. 2000. Tapestry: A dynamic mesh-based display representation for interactive rendering. In

Eurographics Workshop on Rendering, Rendering Techniques, Springer-Verlag, pages 329–340.

SITTHI-AMORN, P., LAWRENCE, J., YANG, L., SANDER, P. V., and NEHAB, D. 2008. An improved shading cache for modern GPUs. In *Proc. of Graphics Hardware*, pages 95–101.

SITTHI-AMORN, P., LAWRENCE, J., YANG, L., SANDER, P. V., NEHAB, D., and XI, J. 2008. Automated reprojection-based pixel shader optimization. *ACM Trans. Graph.*, 27(5):127.

STAMMINGER, M., HABER, J., SCHIRMACHER, H., and SEIDEL, H.-P. 2000. Walkthroughs with corrective texturing. In *Eurographics Workshop on Rendering*, Rendering Techniques, Springer-Verlag, pages 377–388.

STICH, T., LINZ, C., ALBUQUERQUE, G., and MAGNOR, M. 2008. View and time interpolation in image space. *Computer Graphics Forum (Proc. of Pacific Graphics)*, 27(7):1781–1787.

STICH, T., LINZ, C., WALLRAVEN, C., CUNNINGHAM, D., and MAGNOR, M. 2008. Perception-motivated interpolation of image sequences. In *APGV '08: Proc. of Applied perception in graphics and visualization*, ACM, pages 97–106.

SULLIVAN, G. J. and WIEGAND, T. 2005. Video compression from concepts to the h.264-avc standard. In *Proceedings of the IEEE*.

TAWARA, T., MYSZKOWSKI, K., and SEIDEL, H.-P. 2004. Exploiting temporal coherence in final gathering for dynamic scenes. In *Proceedings of the Computer Graphics International*, pages 110–119.

VEDULA, S., BAKER, S., and KANADE, T. 2002. Spatio-temporal view interpolation. In *Proc. of Eurographics workshop on Rendering*, Eurographics Association, pages 65–76.

WALTER, B., DRETTAKIS, G., and GREENBERG, D. P. 2002. Enhancing and optimizing the render cache. In *Eurographics Workshop on Rendering*, Rendering Techniques, Springer-Verlag, pages 37–42.

WALTER, B., DRETTAKIS, G., and PARKER, S. 1999. Interactive rendering using the render cache. In *Eurographics Workshop on Rendering*, Rendering Techniques, Springer-Verlag, pages 19–30.

WARD, G. and SIMMONS, M. 1999. The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Transactions on Graphics*, 18(4):361–368.

WIEGAND, T., SULLIVAN, G. J., BJ ONTEGAARD, G., and LUTHRA, A. 2003. Overview of the h.264-avc video coding standard.

WOOLLEY, C., LUEBKE, D., WATSON, B., and DAYAL, A. 2003. Interruptible rendering. In *SI3D*, ACM Press, pages 143–151.

YANG, L., NEHAB, D., SANDER, P. V., SITTHI-AMORN, P., LAWRENCE, J., and HOPPE, H. 2009. Amortized supersampling. *ACM Trans. Graph.*, 28(5):135.

ZHU, T., WANG, R., and LUEBKE, D. 2005. A GPU accelerated render cache. In *Pacific Graphics (short paper)*.