Image-Based Bidirectional Scene Reprojection

Lei Yang[1] Yu-Chiu Tse[1] Pedro V. Sander[1] Jason Lawrence[2] Diego Nehab[3,4] Hugues Hoppe[3] Clara L. Wilkins[5]
[1]Hong Kong UST [2]University of Virginia [3]Microsoft Research [4]IMPA [5]Wesleyan University

Abstract

We introduce a 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 and efficiency of data reuse since very few pixels are simultaneously occluded in both 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 involves rasterization of scene geometry at each intermediate frame. The second version, our more significant contribution, reduces both shading and geometry computations by performing reprojection using only image-based buffers. It warps and combines the adjacent rendered frames using an efficient iterative search on their stored scene depth and flow. Bidirectional reprojection introduces a small amount of lag. We perform a user study to investigate this lag, and find that its effect is minor. We demonstrate substantial performance improvements (3–4×) for a variety of applications, including vertex-bound and fill-bound scenes, multi-pass effects, and motion blur.

Keywords: real-time rendering, temporal upsampling

Links: DL PDF WEB VIDEO

1 Introduction

Reprojection is a general approach for improving real-time rendering by reusing expensive pixel shading from nearby frames [Scherzer et al. 2011]. It has proven beneficial in popular games. For instance, reverse reprojection [e.g. Nehab et al. 2007; Scherzer et al. 2007] is used in *Gears of War II* to accelerate low-frequency lighting effects, and in *Crysis 2* to antialias distant geometry.

Such data reuse techniques can be broadly categorized based on three separate algorithmic choices:

- *Temporal direction:* Whether shading data is propagated forward, backward, or in both directions in animation time;
- *Data access:* Whether pixels are “pushed” (scattered) onto the current frame, or “pulled” (gathered) from other frames;
- *Correspondence domain:* Whether the motion data (e.g., velocity vectors) used to reproject the samples is defined over the source image domain or over the rendered target.

As reviewed in Section 2, different techniques follow different strategies for each of the choices above (see also Table 1).

In this paper, we present reprojection techniques for temporally upsampling rendered content by inserting interpolated frames between pairs of rendered frames. Borrowing terminology from video compression, we refer to rendered frames as *intra-* or simply *I-frames*, and to interpolated frames as *bidirectionally predicted-* or *B-frames*.

Table 1: *Strategic algorithmic choices in reprojection techniques.*

	Temporal direction	*Data access*	*Correspondence domain*
e.g., Badt [1988]	Forward	Scatter	Source
e.g., Nehab et al. [2007]	Forward	Gather	Target
e.g., Didyk et al. [2010a]	Forward	Hybrid	Hybrid
Our scene-assisted scheme	**Both**	Gather	Target
Our image-based scheme	**Both**	Gather	**Source**

Our approach offers two major contributions: (1) *bidirectional reprojection* which combines samples both forward and backward in time, and (2) *image-based reprojection* which establishes reprojection correspondences based on velocity fields stored in the I-frames.

Temporal direction A fundamental limitation of existing reverse reprojection techniques [e.g. Nehab et al. 2007] 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. This is because such regions must be reshaded from scratch. Since the number of disoccluded pixels varies over time, framerates may fluctuate undesirably. In addition, the entire scene geometry must be processed in order to reshade, incurring significant overhead in complex scenes.

Our *bidirectional* reprojection temporally upsamples rendered content by reusing data from *both* backward and forward temporal directions. This provides two clear benefits:

- *Smooth shading interpolation:* The vast majority of pixels in a B-frame are also visible in *both* I-frames. This lets us fetch shading information from both directions and create an interpolated signal that greatly attenuates the popping artifacts associated with one-sided reconstruction (i.e., sample-and-hold extrapolation). This is particularly important for fast changing shading signals (e.g., dynamic shadows and glossy lighting).
- *Higher, more stable framerate:* Disoccluded regions are extremely rare since they must be occluded in both I-frames. Thus with bidirectional reprojection we can avoid reshading and achieve higher and steadier framerates.

One downside of bidirectional reprojection is that it introduces a lag in the resulting image sequence. This lag is not present in forward-only reprojection schemes. We present a careful analysis of this lag, showing that it is small (less than one I-frame). Moreover, results of a user study we conducted allow us to conclude that it is beneficial to use bidirectional reprojection in a real-time gaming scenario.

Data access Some reprojection techniques use a forward-mapping strategy to scatter shading samples from prior frames into the new frame. However, scatter is difficult to realize efficiently in prevalent graphics systems. Accurate filtering of irregularly scattered samples is also challenging. Like recent reverse reprojection techniques, we use a gather strategy, which simply involves texture lookups into a previously rendered image. The gather operations thus benefit from the accurate, efficient texture-sampling hardware.

Correspondence domain Prior gather-based reprojection techniques require that geometry be rasterized in the target frame to establish a correspondence with source frame pixels. We develop an *image-based* reprojection scheme that instead performs reprojection using only image buffers—without any geometry rasterization—and can therefore accelerate both vertex- and fill-bound scenes. The key idea is to store depth maps and 3D scene flow in the *source* I-frames, so as to enable the B-frame to be reconstructed by an efficient iterative search performed in a fragment shader. Thus, the geometry is rasterized only at the I-frames, and the cost of computing B-frames is simply proportional to the number of framebuffer pixels. Furthermore, we show how to achieve a stable framerate by interleaving the I-frame computation with the much less expensive B-frame processing. 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 spatiotemporal coherence to render animated image sequences more efficiently has been extensively studied in both offline and interactive rendering systems (see [Scherzer et al. 2011] for a recent survey). Here we focus on GPU-based real-time 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; Sitthi-amorn et al. 2008a,b]. Similar approaches have also been applied to amortize expensive sampling computations for improving shading quality [Scherzer et al. 2007; Nehab et al. 2007; Yang et al. 2009; Scherzer et al. 2009; Mattausch et al. 2010]. Finally, a recent technique reuses information from a stack of previously rendered frames to perform spatial upsampling on each frame [Herzog et al. 2010]. In contrast, our approach performs temporal upsampling from both temporal directions.

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; Eisemann et al. 2008]. 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 are 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, by Didyk et al. [2010a], targets high-refresh-rate displays. Similar to our approach, exact forward motion flow of each I-frame is computed when rasterizing the scene, using known animation input of the next I-frame. This flow field is used to compute a forward image warp defined by a coarse grid superimposed over the framebuffer. In contrast, by reconstructing a dense scene flow field that relates each B-frame to both of its adjacent I-frames, our technique produces higher-quality interpolated content and is thus more suitable for applications targeting lower framerates, where individual B-frames are more discernible. A comparison to this method is available in the supplemental material. Didyk et al. [2010b] extend this grid warping method using adaptive grid refinement to achieve better quality in stereo view synthesis. Combining this with bidirectional reprojection is an interesting direction for future work.

In concurrent work, Andreev [2010] presents an efficient framerate up-conversion method that uses motion vectors to perform reverse reprojection in image space. His approach generates an intermediate frame by halving the motion vector in the next I-frame, and uses this motion information to reproject shading result from the previous I-frame. Our approach introduces an iterative search for more accurate reprojection. Andreev prefers reprojection in only the forward temporal direction, aiming for less latency in a third-person game. Bidirectional reprojection produces far fewer disocclusion artifacts and better shading interpolation, which again can be particularly important for interpolating low initial framerates.

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

Pajak et al. [2011] introduce a method for efficiently streaming rendered frame sequences for remote rendering, assisted by exact scene motion data. Their method is based on spatio-temporal upsampling [Herzog et al. 2010] and an efficient edge-image compression scheme, which significantly reduces transmission bandwidth with low computational overhead.

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. For each pixel in a B-frame, the scene flow indicates where to pull shading information from the I-frames.

We 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 $\alpha \in \{\frac{1}{n}, \ldots, \frac{n-1}{n}\}$.

Let the symbol $p = (p_x, p_y)$ denote the 2D coordinates of a pixel in clip space. The third coordinate (depth) is available during geometry rasterization and in the depth buffer Z. Let $\bar{p} = \big(p_x, p_y, Z[p]\big)$ denote its corresponding 3D coordinates. 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 the transformation that maps the surface point $\bar{p}_t$ at time t into the clip space of time t'.

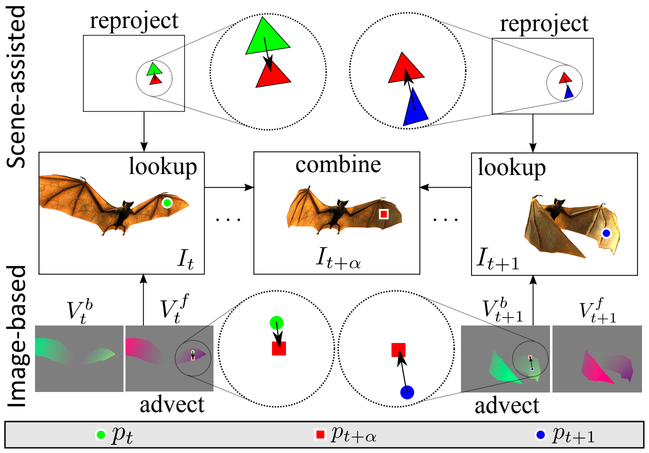


Figure 1: *Overview of scene-assisted and image-based reprojection algorithms for reconstructing a pixel $I_{t+\alpha}[p_{t+\alpha}]$ in a B-frame from its adjacent rendered I-frames. Scene-assisted reprojection computes scene flow during conventional triangle rasterization, while image-based reprojection estimates the flow by iteratively searching the adjacent frames' depth and 3D flow fields.*

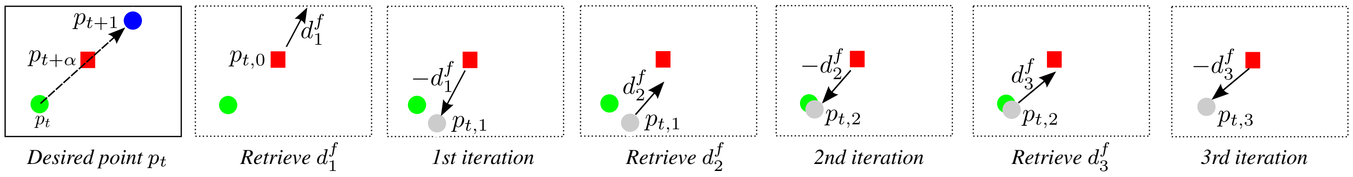


Figure 2: *Three iterations of the search algorithm used to estimate the scene flow at one pixel in a B-frame with respect to the previous I-frame.*

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 exact scene flow between each B-frame and its pair of neighboring I-frames by rasterizing scene geometry (without shading) at each B-frame. (Section 4)
- *Image-based interpolation* computes exact scene flow only between adjacent I-frames and uses a simple iterative search to estimate scene flow between each B-frame and its neighboring I-frames. This sacrifices accuracy in the interpolated frames, but allows geometry to be rasterized just in the I-frames. (Section 5)

We next describe these two strategies in detail. Section 9 presents results along with a comparison.

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 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 standard 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 the rendering of 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 clip space 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, 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 use the closest-on-miss approach. This strategy maps well to the SIMD architecture of modern graphics hardware since it avoids any conditional execution of the expensive surface shader and resulting 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. The idea is to augment the I-frame buffers with information about the 3D scene flow between adjacent I-frames, and use that to drive an image-based interpolation process. More specifically, while F_t is rendered we compute and store a *forward flow field* V_t^f that 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 of F_{t+1} that 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 (see [Nehab et al. 2007]).

I-frame At time t, we first render V_t^f. Then, we render the buffers I_{t+1}, Z_{t+1}, and V_{t+1}^b associated with time $t+1$ in a single rendering pass using multiple render targets. Note that buffers I_t and Z_t, which will also be needed to reconstruct 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 interpolate their colors. As described next, we estimate these positions using an iterative search algorithm that can be efficiently implemented on graphics hardware and then discuss some useful optimizations.

5.1 Iterative search

For each pixel $p_{t+\alpha}$ in B-frame $F_{t+\alpha}$, we need to compute the corresponding neighboring locations $p_t = \pi_{t+\alpha\to t}(\bar{p}_{t+\alpha})$ in I-frame F_t and $p_{t+1} = \pi_{t+\alpha\to t+1}(\bar{p}_{t+\alpha})$ in I-frame F_{t+1}. 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 instead approximate these positions using a greedy search directed by the forward and backward flow fields.

Forward direction We assume that the pixel coordinate $p_{t+\alpha}$ and its corresponding coordinate p_t in I-frame I_t are related to one another according to the fractional displacement $\alpha V_t^f[p_t]$ along the forward flow field (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 F_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 F_t may also map to $p_{t+\alpha}$ but may be occluded in this frame.

We estimate p_t using a simple search (see Figure 2). We start with

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

and iteratively compute

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

To make the algorithm efficient on a SIMD architecture, we always terminate the search after a fixed number m of iterations ($m = 3$ for all of the results in this paper). 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,

$$p_{t+1,0} = p_{t+a} \; , \tag{8}$$

$$p_{t+1,i} = p_{t+\alpha} - d_i^b \; , \tag{9}$$

$$d_i^b = (1-\alpha) \; V_{t+1}^b[p_{t+1,i-1}].xy \; . \tag{10}$$

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 are seldom 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:

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

   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 that both errors exceed the tolerance, we apply the blending procedure in step 1 to the solutions of both searches.

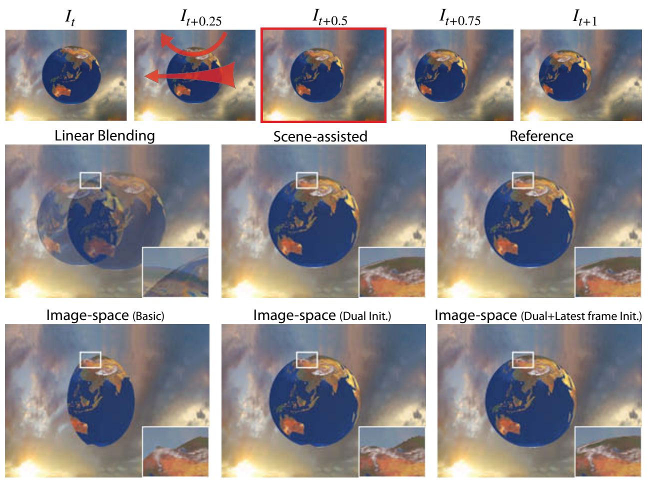


Figure 3: *Bidirectional reprojection on an animated scene that involves translation, rotation, and scaling.*

5.3 Additional search initializations

Although the search described above produces good matches in many cases, there are situations where it fails to identify a correct correspondence. We have developed several alternative initialization strategies that improve performance in other common cases. In practice, we always perform these in addition to the default initialization strategy described previously and use whichever solution has the smallest reprojection error. In Section 6, we discuss how our method can be adapted for scenes that include objects that are simply not suitable for image-based bidirectional reprojection.

Dual initialization One difficulty occurs 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 3.

In these situations, it is preferable to use a different starting point for each of the two iterative searches. Specifically, we initialize the search in one I-frame using the velocity vector retrieved from the opposing I-frame:

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

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

This avoids having the search initially miss the object and allows converging to the proper displacement. We follow the same iterative procedure described previously with these alternative starting points.

Latest-frame initialization Another useful observation is that for all but the first B-frame within a single interval, we can exploit the scene's natural temporal coherence and initialize the search at each pixel with 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 the neighboring B-frames.

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

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

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:

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

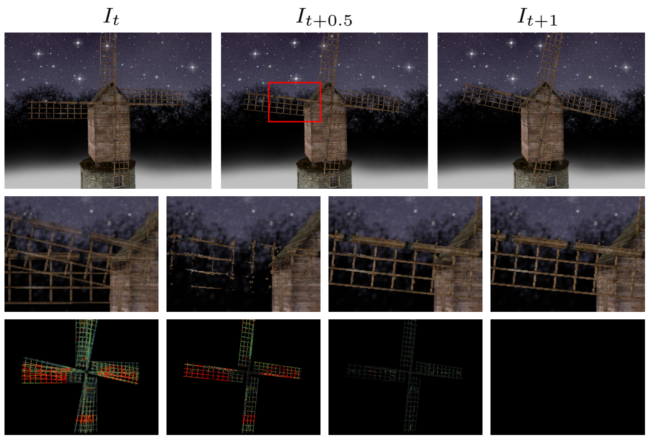


Figure 4: *Upsampling a scene with fast moving thin geometry using: (a) Linear blending; (b) Image-based interpolation; (c) Image-based interpolation on the entire scene and scene-assisted interpolation on the blades; (d) Image-based interpolation on the entire scene except the blades and standard rendering of the blades. The error images in the last row use the same color map as in Figure 7–9.*

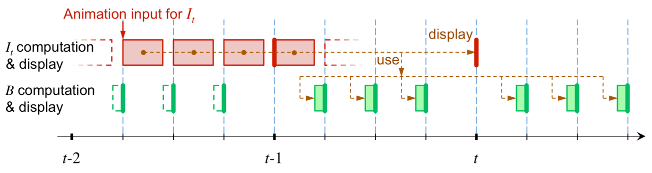
where $d_i^{\prime b}$ is the offset computed in the previous B-frame at time $t + \alpha'$. To obtain even better results, we consider all of the offsets within a small fixed neighborhood around the pixel and use whichever has the smallest depth. This proved to be particularly helpful near depth discontinuities. For example, consider the pixels near the north pole of the globe in Figure 3. Using motion vectors from nearby pixels allows the algorithm to "find" the globe in previous frames, and thus initialize the search using a motion vector that is consistent with the underlying motion of this object.

6 Limitations and interoperability

Although our geometry- and image-based algorithms generate high quality output for a wide range of practical cases (see Section 9), there are scenes for which they may produce unacceptable errors. As identified by early work on reprojection [Nehab et al. 2007], dynamic shading effects such as highlights, shadows, transparency, and reflections can lead to signals that move rapidly relative to object surfaces, and any such drastic motions may lead to popping artifacts in the shading signal with sample-and-hold extrapolation. On the other hand, since our approach interpolates between two I-frames it produces linearly blended shading results, which are far less objectionable (see the comparison in Section 9).

In addition, our image-based algorithm is prone to make errors in the interpolated B-frames wherever the local search fails to compute a correct correspondence between I-frames. Although these cases are normally rare, they may become noticeable on thin and fast-moving geometry like the windmill blades in Figure 4.

An important benefit of our interpolation techniques is that they can be used together and can even be integrated with traditional rendering. Figure 4 illustrates this type of interoperability. Our image-based reprojection algorithm struggles to properly track and interpolate the motion of the thin windmill blades (Figure 4(b)). However, it is possible to perform an additional scene-assisted reprojection pass on just the blades to patch the missing pixels in the image-based interpolated frame (Figure 4(c)). Here the scene depth generated by the image-based pass is stored in the Z-buffer, which is then used in the scene-assisted pass to ensure correct depth culling. The I-frames are shared between the two approaches and there is no additional storage overhead. Using the scene-assisted approach for only the thin geometry helps to reduce the amount of geometry processing overhead. Note that the scene shown in



Animation input for I_t
I_t computation & display
B computation & display
display
use
t-2
t-1
t

Figure 5: *The partitioned computation of the I-frame I_t takes place within the time interval $[t-2+\frac{1}{n}, t-1+\frac{1}{n}]$. That I-frame gets reprojected into the B-frames in the interval $(t-1, t+1)$.*

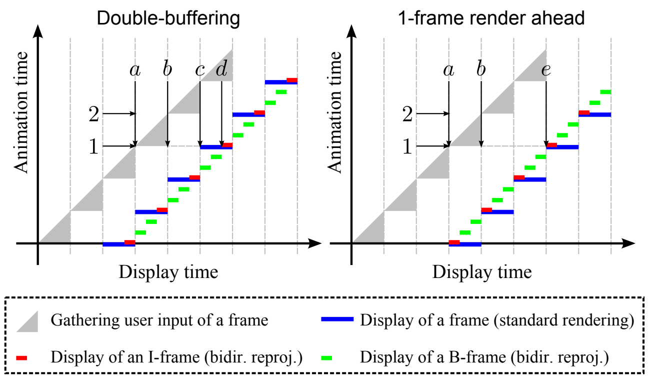


Figure 6: *Analysis of lag under different rendering scenarios: standard double-buffering and a 1-frame render-ahead queue (or flip-queue). Additional lag introduced by the partitioned variant is only $\frac{n-1}{n}$ time steps in the former, and vanishes in the latter.*

Figure 4 demonstrates a rather challenging case with some "double miss" regions behind the moving blades. Our image-space algorithm attempts to find the closest patch for these regions, which may lead to some visual error in shading (see the error image in Figure 4(c)). Fortunately, this issue can be avoided by applying image-space interpolation only on the scene elements behind the blades and then performing conventional rendering on the blades, using the same Z-buffer bridging approach described above (Figure 4(d)). The main difference here is that the I-frames contain no fast moving thin elements, thereby preventing the complex visibility changes that cause double misses.

7 Partitioned rendering and lag

Partitioned rendering The expensive computation of I-frames (relative to the cost of B-frames) results in uneven rendering load. To address this problem, we partition the computation of I_t over the several consecutive B-frames in the interval $[t-2+\frac{1}{n}, t-1+\frac{1}{n}]$, as shown in Figure 5. The rendering of I_t must be completed by time $t-1+\frac{1}{n}$ because it is needed for the rendering of that B-frame. Amortization is achieved by partitioning the scene rendering tasks as evenly as possible and interleaving these tasks with the rendering of B-frames. This process is application-dependent and may require profiling the cost of all render tasks in a frame. In a modern game engine which typically issues hundreds to thousands of draw calls, amortization can be achieved by dividing the draw calls into groups with roughly uniform render cost, launching each group in individual I- and B-frames and accumulating the results on the appropriate render targets. This is also compatible with deferred renderers in which small chunks of post-processing tasks dominate the entire shading cost.

With more flexible hardware and/or drivers, it may be possible to render several B-frames consecutively into a queue of framebuffers, which could then be sent asynchronously to the display. Scene partitioning would then become unnecessary.

Analysis of lag In order to illustrate the additional lag introduced by our method, we examine two standard scenarios in Figure 6.

In both cases, we assume that rendering an I-frame takes slightly less than one time step, and that vertical synchronization (VSync) is enabled. The vertical axis represents animation time, the time at which input was gathered and the command buffer is sent to the graphics card for rendering. In contrast, the horizontal axis represents display time, the time at which the frame corresponding to a certain animation time is shown to the user. We now analyze the additional lag, relative to standard rendering, that is introduced by bidirectional reprojection, using the partitioned rendering strategy described above.

When double-buffering is used, any feedback concerning events in the interval $[1, 2)$ can only contribute to rendering after the refresh at b, since the GPU is busy before that (figure shows the worst case). Rendering completes right before c and the frame depicting animation time 1 (in blue) is traditionally shown to the user at time c. In contrast, our method presents the first B-frame (in green) that combines information from the frames depicting animation times 1 and 2 at display time c. After $n - 1$ B-frames, we finally show the I-frame (in red) for animation time 1 at display time d. The additional lag is therefore $\frac{n-1}{n}$ time steps.

A common alternative that is also widely employed in the industry is to use a 1–3 frame render-ahead queue provided by the graphics driver in order to maintain more stable framerates. Interestingly, when a 1-frame render ahead queue is used in conjunction with our approach, the additional lag due to our method vanishes. Traditional rendering has to wait for an entire additional I-frame to finish rendering in about one time step before it can display the frame showing animation time 1 at e. Since we can render B-frames quickly, our method waits only an additional $\frac{1}{n}$. Therefore, both methods present the same information at time e, i.e., there is no additional lag.

Given this analysis, we should expect the amount of lag perceived by users exposed to our method to be similar to what they perceive in a traditional setting. Indeed, results of the user study we describe in Section 8 support this claim. Another concern is whether the B-frames really represent an intermediate animation time, as shown in Figure 6. We believe so, since both motion *and* colors are interpolated. Our user study also supports this claim.

8 User study

In this section, we analyze the effects on user performance and experience of the tradeoff between the higher framerates and increased latency associated with our technique. Although smoother animations allow users to better track object motion—thereby helping with interactive tasks—increasing feedback latency may well counteract these gains by forcing users to act according to longer predictions into future motion. To answer such questions, we conducted a user study in which we measured the performance and preferences of a group of subjects interacting with a custom-made game of skill.

Game design The game shows a number of red and green spheres falling downward, while bouncing against each other and against the bottom of the screen. Each game match lasts for 45 seconds. The goal in each match is to click over as many green circles as possible, while at the same time avoiding red circles and ineffective clicks. Circles disappear when clicked. We used this simple action game that requires reflex skills because rendering latency can be a hindrance in this scenario.

The game is played in 7 different rendering modes. The first three are traditional rendering modes at 60, 30, and 15fps. Three additional modes render at 60fps, but artificially introduce increasing amounts of lag: 50ms, 100ms, and 200ms. The remaining mode implements our bidirectional reprojection technique with partitioned rendering and upsampling from 15 to 60fps. The idea is that using such a low framerate as a starting point should help reveal any effect of lag.

Hypotheses The important hypothesis here is that the lag due to our method will not cause the player's *objective performance* to suffer relative to standard 15fps rendering. Any improvements would be welcome, but certainly unnecessary given our assumption that 15fps is the best that can be achieved without our technique. Naturally, we also hypothesized that the *subjective* assessment would show that users generally prefer the smoother experience provided by our method when compared against standard 15fps rendering.

The study Subjects were 33 predominantly male graduate and undergraduate students who accepted taking part in the study when offered compensation (chocolate) or the chance of winning a prize (a fancy flash-memory drive).

Gaming session Each subject played 21 matches, divided into two rounds. The first round, containing 7 matches (one in each mode), was used to expose users to each game mode in random order. No scores were recorded. Measurements were recorded during the second round, which contained 14 matches (two in each mode), presented in random order.

For each match, we recorded the number of green, red, and missed clicks. These results allow us to measure objective performance under different rendering modes.

After each match, participants were asked to rate their agreement with a variety of statements, on a scale from 1 (*strongly disagree*) to 7 (*strongly agree*). The statements were "I enjoyed playing this game," "This game was difficult to play," "This game was responsive to my input," and "The animation was smooth." These measurements allow us to assess subjective preferences.

Results We ran repeated-measures ANOVAs to examine differences between rendering modes, and they were all significant ($p < .001$)[1]. We then ran simple contrasts to compare each mode against our method. Results can be seen in Table 2, and are summarized below.

Objective measures

- *Green and red spheres hit*. Comparing our method against 15fps did not produce statistically significant results ($F(1, 32) = .10$, $p = .75$)[2]. Users performed better in the 60fps, 30fps, and 50ms modes, and performed worse when playing with 100ms and 200ms of lag.
- *Misses (clicks over the background)*. Subjects made fewer useless clicks when playing with our method than 15fps, 100ms, and 200ms of lag. They made even fewer mistakes when playing the 60 and 30fps modes. Results of 50ms of lag were only marginally better ($p = .058$).

Subjective measures

- *I enjoyed playing this game*. Although the favorite game was the standard 60fps, participants indeed reported enjoying our method more than 15fps, 100ms lag, and 200ms lag. Comparisons against 30fps and 50ms of lag were not statistically significant.
- *This game was difficult to play*. Participants felt that our method was easier to play than 15fps and 200ms lag, but thought 60fps and 30fps were even easier. Comparisons against 50ms and 100ms lags were not statistically significant.
- *This game was responsive to my input*. We were unable to differentiate statistically between the perceived responsiveness of our method and 15fps (as we hoped for) or 100ms of lag. Subjects thought our mode was more responsive than 200ms of lag, but less responsive than 50ms of lag, 60fps, 30fps (the latter two as expected).

[1] We consider a result statistically significant whenever $p < .05$.

[2] We will not report the remaining F values in the interest of brevity.

Table 2: *Results of the user study. We report means (M) and standard deviations (SD) for each rendering mode, and p values for each simple contrast. Mean figures marked in red compare favorably against our method. On the other hand, our method outperforms those marked in blue. Unmarked mean values are not statistically different from the results of our method ($p > .05$) according to the simple contrasts.*

Mode	Ours		60fps (infeasible)			30fps (infeasible)			15fps (feasible)			50ms lag			100ms lag			200ms lag		
Statistics	*M*	*SD*	*M*	*SD*	*p*	*M*	*SD*	*p*	*M*	*SD*	*p*	*M*	*SD*	*p*	*M*	*SD*	*p*	*M*	*SD*	*p*
Green hits	31.64	7.37	**43.79**	11.07	.000	**41.65**	9.86	.000	31.89	6.82	.754	**35.47**	7.24	.000	**29.85**	6.98	.045	**24.24**	5.72	.000
Red hits	3.44	2.91	**1.92**	1.53	.001	**2.20**	1.76	.003	3.32	1.75	.818	3.11	2.84	.345	**4.33**	2.78	.018	**6.68**	3.55	.000
Misses	22.01	1.45	**15.68**	8.34	.000	**19.32**	9.40	.006	**25.42**	12.63	.010	20.38	10.74	.058	**25.11**	13.73	.007	**31.59**	17.16	.000
Enjoy	4.38	1.14	**5.42**	1.15	.000	**4.65**	1.07	.107	**2.61**	1.18	.000	4.47	1.08	.589	**3.77**	1.01	.000	**2.88**	.977	.000
Difficult	4.26	1.01	**3.24**	1.29	.000	**3.85**	1.09	.010	**4.80**	1.27	.019	4.00	1.05	.114	4.58	1.17	.090	**5.00**	1.05	.002
Responsive	4.03	1.09	**5.74**	1.08	.000	**5.18**	.98	.000	3.71	1.25	.139	**4.59**	1.07	.013	3.68	1.27	.097	**2.45**	1.19	.000
Smooth	5.36	1.22	5.67	1.24	.074	**3.71**	1.25	.000	**1.80**	.90	.000	5.38	1.21	.922	5.06	1.31	.096	**4.83**	1.19	.006

Table 3: *Breakdown of B-frame average rendering times (in milliseconds) at* 1024×768 *for each of the three test scenes. Each B-frame rendering cycle first renders an allotted partition of an I-frame as described in section 7 ("Render"), and then performs bidirectional reprojection to interpolate its adjacent I-frames ("Interp").*

Scene	Original	Scene-Assisted			Image-Based		
		Render	*Interp*	*Total*	*Render*	*Interp*	*Total*
Walking	67.0	20.0	3.3	23.3	23.1	2.6	25.7
Terrain	126.6	33.0	87.9	120.9	42.4	2.2	44.6
Head	28.3	7.6	0.8	8.4	7.9	2.0	9.9

- *The animation was smooth.* Subjects rated our method as producing smoother animations than both 30fps and 15fps (as expected). Interestingly, users also preferred our method when compared with 200ms lag, probably out of frustration with the large amount of lag. Comparisons against 60fps, 50ms, and 100ms of lag were not statistically significant.

Discussion Our method did in fact fair slightly better than 15fps in objective measurements. Subjective measures were more salient, and showed significant favor toward our method. Finally, as far as perception of lag is concerned, the experiments showed that our method is positioned somewhere between 50ms and 100ms. Recall that 60fps and 30fps are not viable alternatives under our assumptions, since our computational budget only allows for 15fps. This is why our method is advantageous.

Given that the effect of lag would be even smaller when upsampling from higher framerates (say, from 30fps to 60fps), we are confident our technique would be even more suitable for these scenarios.

The additional lag present in bidirectional reprojection with double-buffering has a two-fold effect in a first- or third-person shooter game. Not only are the scene elements affected by the display lag, but so is the aiming action. Although the additional lag is small, this may potentially make it more challenging to aim in some fast action games that require instantaneous response to abrupt target moves. Our user study does not explore this aspect. We believe such effects should be evaluated on a case-by-case basis.

9 Results

In this section we discuss different usage scenarios for bidirectional reprojection and analyze their quality/speed tradeoffs. Our results were generated using an Intel Core Duo 3GHz CPU with 2GB of RAM and an NVIDIA GeForce 8800 GTX graphics card.

Figures 7–10 present results for both scene-assisted and image-based bidirectional reprojection for several different scenes. 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 scene (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 terms of SSIM [Wang et al. 2004]) compared to the reference images over a given animation sequence. We also show close-ups and difference images for an intermediate B-frame. Images for all three intermediate B-frames $I_{t+0.25}$, $I_{t+0.5}$, and $I_{t+0.75}$ along with their MSE error graphs are available as supplemental material. Note that one out of every four frames is an I-frame and therefore has no error. Hence a periodic pattern is visible in the quality measurement graphs. Note that the graphs are displayed with respect to wall clock time. Therefore, the temporal locations of the I-frames in the different techniques are not synchronized due to framerate differences. However, for the image comparisons, we did synchronize the temporal locations to provide the fairest comparison of visual quality.

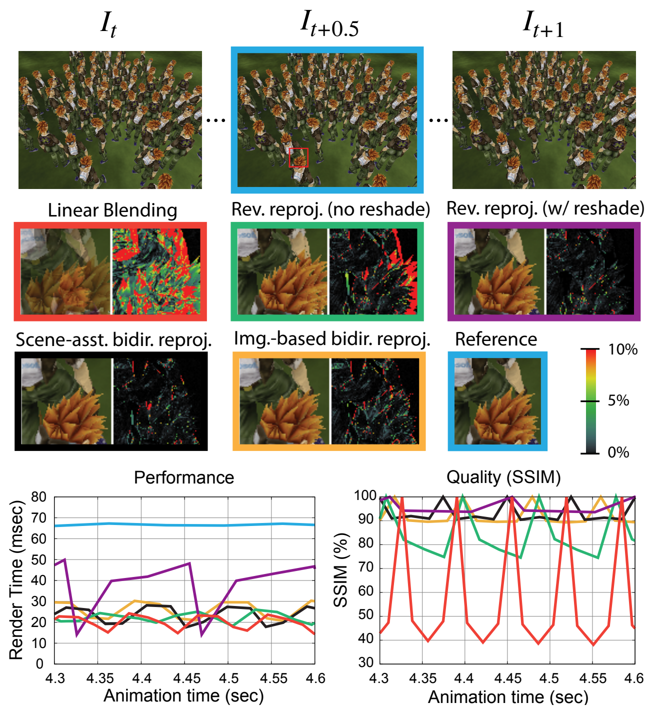


Figure 7: *Results of our algorithm on the walking scene. The color of each plot lines matches the inset frame of the corresponding method. The closeups show error images.*

In some test scenes, 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.

Table 3 shows a breakdown of time taken at each B-frame. Note that the time for rendering and shading the allotted portion of an I-frame (as described in Section 7) is the dominating cost. The interpolation and display of the B-frame using either geometry-assisted or image-based reprojection is very fast. The exception is the terrain scene, which is heavily vertex bound, therefore making geometry-assisted reprojection infeasible. The image-based interpolation is a screen-space technique and therefore is independent of geometric scene complexity. It takes just 2-3 milliseconds to interpolate and display the B-frame. Note that the total time to render an I-frame using partitioned rendering is slightly larger than using traditional rendering, due to additional overdraw and the overhead of issuing draw calls with instancing. The *Render* time of the image-based method also includes the time for generating motion flow buffers, hence they are slightly larger than that of the scene-assisted method.

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

Fill-bound scenes The *walking* scene of Figure 7 is an example of a fill-bound scene with characters moving over a floor shaded by an expensive procedural noise function. Fill-bound scenes are common in real-time applications (e.g., computer games), and bidirectional reprojection significantly reduces their rendering cost by reducing the number of expensive pixels that need to be processed. The bottom row of Figure 7 shows quality and performance results of bidirectional reprojection for a representative animation segment. Note that it is nearly 3× faster 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 comes 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 8, most of the rendering budget is consumed by 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× 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 supplemental 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 in vertex processing, both 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 could 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.

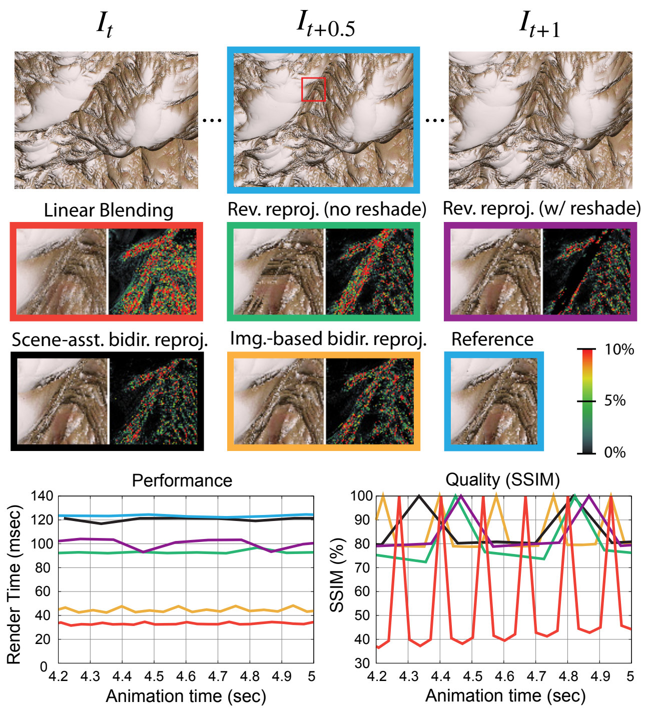


Figure 8: *Results of our algorithm on the terrain scene. The lines in the plots are colored according to the color of the frame around the insets of the corresponding method.*

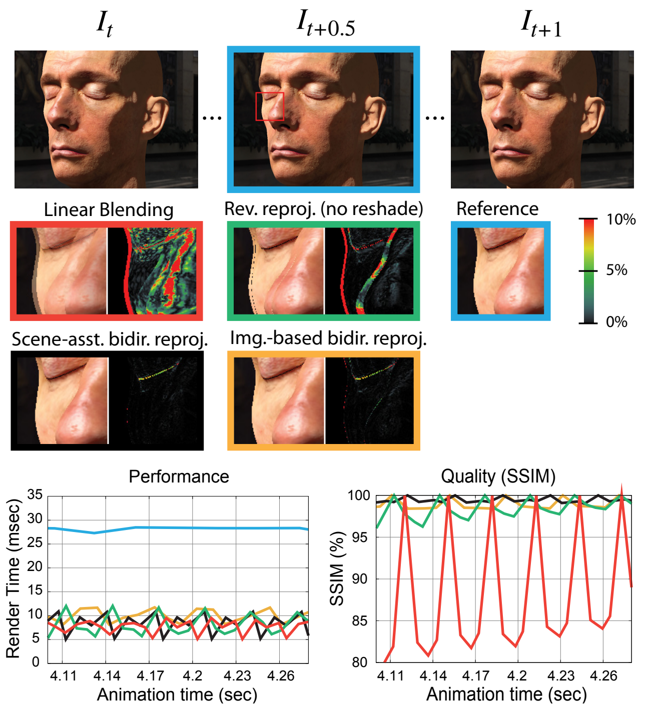


Figure 9: *Results of our algorithm on the NVIDIA human head scene. The lines in the plots are colored according to the color of the frame around the insets of the corresponding method.*

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* example in the Figure 9 is such an example. It uses

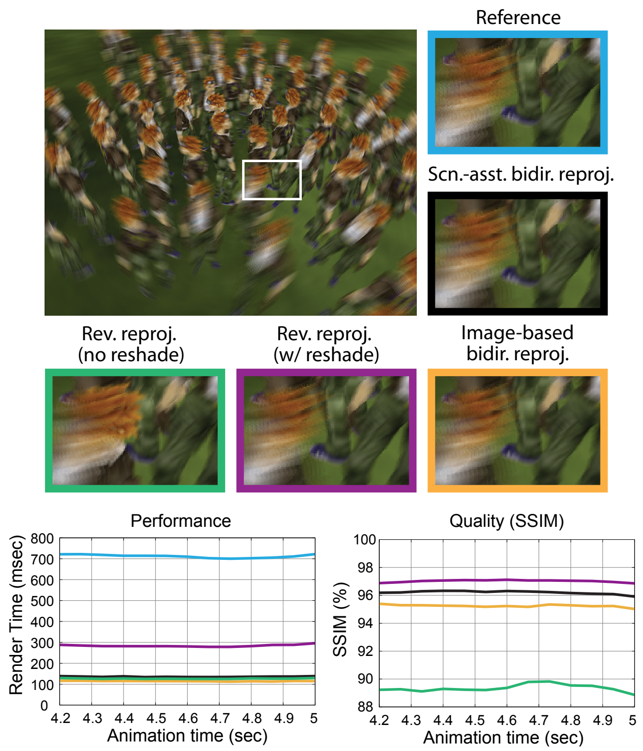
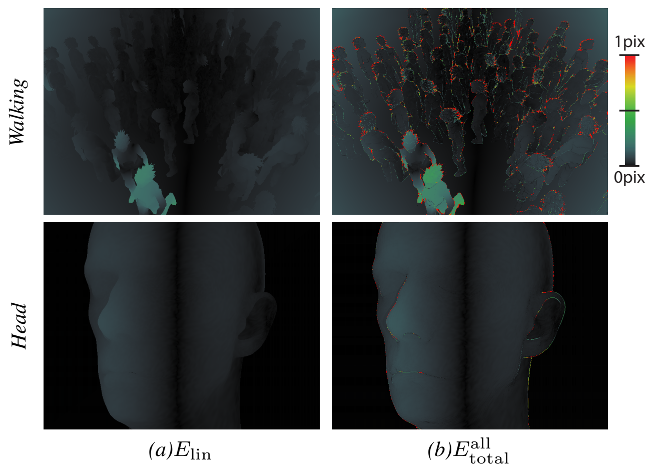


Figure 10: *Results of our algorithm on the walking scene with motion blur.*

a sum-of-Gaussians formulation of subsurface scattering which is 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 needs 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, and moreover 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 scene.

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 10), 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) tradeoff between speed and quality in this case, as illustrated by these graphs. Single-direction reprojection with reshading achieves better results, but is significantly slower since it has to reshade on cache misses. Single-direction reprojection without reshading is fast, but again suffers from significant artifacts at disoccluded regions.

Image-based reprojection error analysis There are two potential sources of error in our imaged-based reprojection technique. One is the error E_{lin} due to the assumption that surface points follow linear trajectories between I-frames. To measure E_{lin}, we can use the motion vectors between two I-frames to compute where each I-frame pixel would appear in a B-frame (under the linear motion assumption) and compare it with its exact position, which can be obtained with scene-assisted reprojection. This was the procedure followed to generate the results in Figure 11 *(a)*. For the challenging



Scene	E_{lin}			$E^{\text{all}}_{\text{total}}$			$E^{\text{hit}}_{\text{total}}$		
	$<\frac{1}{10}$	$<\frac{1}{2}$	<1	$<\frac{1}{10}$	$<\frac{1}{2}$	<1	$<\frac{1}{10}$	$<\frac{1}{2}$	<1
Walking	97.6	100	100	89.9	97.5	98.2	94.3	99.6	99.8
Head	99.9	100	100	98.6	99.9	99.9	98.7	100	100

Figure 11: *Reprojection error in the head and the walking scene: (a) due to the linear motion assumption, and (b) in the result obtained with our image-based iterative searching algorithm. Both errors measured in the unit of a pixel size. The table shows the percentage statistics of different error range.*

walking scene, which contains fast nonlinear animation and camera rotation, E_{lin} is below 0.5 pixels everywhere, and is typically much smaller than this amount. Similar results apply to the human head scene. We conclude from these results that E_{lin} is not a significant source of error.

The remaining source of error is our iterative search strategy, which may fail to converge to the correct position under our linear assumption. Since there is no ground-truth in this case, we cannot in general isolate this source of error. Instead, we measure the total reprojection error E_{total}, which combines the influence of E_{lin} with the potential failure of our iterative search. To obtain E_{total}, we measure the distance between the correspondences the iterative search finds for each B-frame pixel in the I-frames and the ground-truth correspondences reported by the scene-assisted method. Results of this analysis are shown in Figure 11 *(b)*. E_{total} is very small except near silhouettes, where correspondences may not exist and where depth discontinuities interfere with our bilinear reconstruction kernel. (These issues affect even the scene-assisted method and traditional reverse reprojection.) To analyze the error more carefully, we report statistics for E_{total} over all pixels ($E^{\text{all}}_{\text{total}}$) and also restricted to pixels whose correspondences actually exist ($E^{\text{hit}}_{\text{total}}$). Note the significant drop in $E^{\text{hit}}_{\text{total}}$ relative to $E^{\text{all}}_{\text{total}}$.

Finally, note that in addition to being very small, errors are hardly noticeable at high frame rates, as seen in the supplemental video. Also, the error in the interior regions is locally consistent, which prevents visible distortions of detailed textures (see Figure 7).

Improved shading interpolation Traditional reverse reprojection only reuses the shaded value from an earlier time frame. This "sample and hold" strategy leads to larger shading errors in the presence of dynamic lighting and shadows as shown in Figure 12. In contrast, bidirectional reprojection always reshades the entire scene anew for each I-frame and temporally interpolates at the B-frames. As a result, it not only reduces the shading error, but also provides a smoother animation with less popping artifacts. The supplemental video contains an interactive version of this comparison.

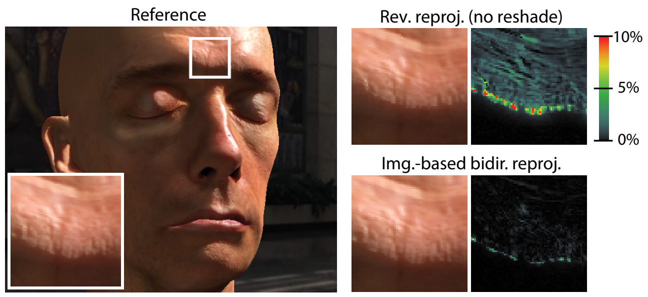


Figure 12: *Comparison of our interpolation method with traditional single-directional reprojection in the presence of dynamic lighting and shadow.*

10 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 following 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 to handle semi-transparent geometry.

Acknowledgements

We thank Piotr Didyk for providing models and results for comparison, and NVIDIA and XYZRGB Inc. for providing the "human head" shader and model. This research was partially supported by NSF CAREER Award CCF-0747220, Hong Kong RGC GRF grants #619008 and #619509, and an INST grant from FAPERJ.

References

ANDREEV, D. 2010. Real-time framerate up-conversion for video games. In *SIGGRAPH 2010 Talks*.

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

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

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

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

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. *Computer Graphics Forum*, 29(2).

DIDYK, P., RITSCHEL, T., EISEMANN, E., MYSZKOWSKI, K., and SEIDEL, H.-P. 2010. Adaptive image-space stereo view synthesis. In *Vision, Modeling, and Visualization*.

EISEMANN, M., DE DECKER, B., MAGNOR, M., BEKAERT, P., DE AGUIAR, E., AHMED, N., THEOBALT, C., and SELLENT, A. 2008. Floating textures. *Computer Graphics Forum*, 27(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.

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 *Proc. Symposium of Interactive 3D Graphics*, pages 7–16.

MATTAUSCH, O., SCHERZER, D., and WIMMER, M. 2010. High-quality screen-space ambient occlusion using temporal coherence. *Computer Graphics Forum*, 29(8):2492–2503.

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

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.

PAJAK, D., HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., and SEIDEL, H.-P. 2011. Scalable remote rendering with depth and motion-flow augmented streaming. *Comp. Graph. Forum*, 30(2).

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

SCHERZER, D., SCHWÄRZLER, M., MATTAUSCH, O., and WIMMER, M. 2009. Real-time soft shadows using temporal coherence. *LNCS (Proc. ISVC)*, 5876:13–24.

SCHERZER, D., YANG, L., MATTAUSCH, O., NEHAB, D., SANDER, P. V., WIMMER, M., and EISEMANN, E. 2011. A survey on temporal coherence methods in real-time rendering. In *Eurographics State of the Art Reports*.

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

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.

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 *Proc. Appl. Percept. Graph. Visul.*, 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*.

VEDULA, S., BAKER, S., and KANADE, T. 2002. Spatio-temporal view interpolation. In *Eurograph. Workshop on Rendering*.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., and SIMONCELLI, E. P. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Proc.*, 13(4):600–612.

WIEGAND, T., SULLIVAN, G. J., BJ ONTEGAARD, G., and LUTHRA, A. 2003. Overview of the H.264-AVC video coding standard. *IEEE Trans. Circ. Syst. Video Tech.*, 13.

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