EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

Volume 31 (2012), Number 2

Iterative Image Warping

Huw Bowles[1] Kenny Mitchell[2] Robert W. Sumner[2] Jeremy Moore[1] Markus Gross[2,3]

[1]Disney Interactive Studios [2]Disney Research Zurich [3]ETH Zurich

Abstract

Animated image sequences often exhibit a large amount of inter-frame coherence which standard rendering algorithms and pipelines are ill-equipped to exploit, limiting their efficiency. To address this inefficiency we transfer rendering results across frames using a novel image warping algorithm based on fixed point iteration. We analyze the behavior of the iteration and describe two alternative algorithms designed to suit different performance requirements. Further, to demonstrate the versatility of our approach we apply it to a number of spatio-temporal rendering problems including 30-to-60Hz frame upsampling, stereoscopic 3D conversion, defocus and motion blur. Finally we compare our approach against existing image warping methods and demonstrate a significant performance improvement.

Categories and Subject Descriptors (according to ACM CCS): I.3.m [Computer Graphics]: Miscellaneous—Image-based rendering

1. Introduction

In computer graphics significant coherence is exhibited across frames of an animation (*temporal coherence*) and across nearby views of a scene (*spatial coherence*). Current rendering pipelines recompute each frame from scratch, resulting in a large amount of repeated work in processing the scene geometry and in performing the light simulation necessary to shade the surfaces. This inefficiency imposes severe constraints on the visual fidelity of real-time applications in which rendering budgets are measured in mere milliseconds. Consequently enormous time and effort are invested in optimization of algorithms and assets which increases rendering pipeline complexity, complicates content creation workflows and drives up production costs.

Although previous work includes a number of approaches to re-use information across frames to exploit this redundancy, they are yet to meet the rigorous requirements of production scenarios and have therefore eluded widespread adoption. These methods incur a heavy geometry processing cost or require significant changes to rendering pipelines and workflows.

In this work we argue for the reuse of shading information across frames through image warping techniques. To transfer this information efficiently we introduce a novel image warping algorithm based on fixed point iteration. We describe a heuristics-based variant of our approach that runs with minimal performance overhead. Additionally from the mathematical properties of fixed point iteration we derive a local convergence criterion which informs a robust adaptive variant of our approach which requires a little more computation time but still a fraction of the time of comparable methods. Our approach can be attached to existing rendering pipelines with minimal side effects, which we demonstrate by installing our warping kernel into an existing AAA console game to post convert it stereoscopic 3D and to upsample its frame rate from 30 to 60Hz. Finally we compare our method to existing work and demonstrate a substantial performance improvement without loss of quality.

2. Background

In this section we review the available options for exploiting frame to frame coherence. *Reverse reprojection caching* (RRC) [NSL*07, SaLY*08] rasterizes the scene from the target viewpoint and computes at each pixel the inverse mapping from target to source image frame. Although the authors report reduced fragment shader cost, the scene geometry needs to be reprocessed once or twice per reprojection which is prohibitively expensive for many production scenes. Additionally, shader authoring is complicated by the need to write variants of the vertex and fragment programs of each shader instrumented with the required reprojection code.

For these reasons we focus on methods that are decoupled from the scenes geometry such as the *render cache* introduced by Walter et al. [WDP99, WDG02], which caches shading points as samples in 3D space and reuses them across frames. However this system requires substantial changes to the graphics pipeline (the presented system relies on ray tracing to populate the cache) and it is not clear how this method could be implemented efficiently on rasterization architectures.

Instead of caching data as shading points, *image-based rendering* (IBR) methods use images as the cache primitive, and use image warping techniques to construct the required viewpoint. Some methods render from a large database of images that sample a global plenoptic function, i.e. the radiance passing through a scene [LH96, GGSC96]. Others store image representations of individual elements in the scene [Max96, SS96, SLS*96, SGHS98]. The *Talisman* system was proposed as a hardware implementation of the latter [TK96]. Due to their modest computational requirements these approaches are well suited to solving the rendering efficiency problem. However, these approaches usually assume that the incoming radiance from surfaces in the scene is static which may be too restrictive for many scenarios.

This issue is mitigated in *post-rendering warping* which is the application of image warping to the reprojection of rendered images to nearby viewpoints in space and time. While post-rendering warps also assume that the radiance function is static, for small reprojection offsets the resulting error tends to be small. Existing image warping methods can be categorized by their data access patterns. *Forward warps* scatter information for the source frame into the target frame, which can be implemented on graphics hardware in the following ways. Each pixel can be splatted as individual primitives [Wes90, ZPvBG02] which is conceptually simple but efficiently and accurately filtering irregularly scattered samples is challenging on GPU hardware. Another approach is to connect the pixels with triangles and rasterize them into the target view [MMB97], which prevents issues like holes appearing in the warped image. However both approaches result in a heavy geometry processing workload (over 1M primitives per warp at standard resolutions). While this dense warp geometry can be reduced by using a coarse regular grid [DER*10] or by adaptively grouping pixels into coherent blocks [CW93, DRE*10], such methods are an approximation of the true warp.

Inverse warps reverse the data access pattern, i.e. they gather information from the source frame to fill the target frame. Since a closed-form expression for the inverse warp is often not available, special heuristics for the warp type are used to obtain an approximate solution [KTI*01, And10, SKS11], or a manual search is conducted through the source image [McM97, Mar98]. A more efficient way to perform this search is to employ an iterative method called *fixed point iteration*. Such an approach has recently been applied to the inversion of computed tomography data by Chen et al. [CLC*08]. However they address the limited case for which a global inverse warp function exists (i.e. there are no overlaps/folds in the warp), and is not applicable to the warps considered in this work. We previously relaxed this restriction [Bow10] and illustrated the behavior of the iteration in the presence of disocclusions in the context of stereoscopic 3D post-conversion and temporal upsampling. The temporal upsampling approach from Yang et al. [YTS*11] also employs an iteration equivalent to fixed point iteration but they do not identify the iteration as such or consider its convergence properties further. In this work we study the behavior of the iteration in detail and establish a general framework for image warping using fixed point iteration.

3. Method

We aim to warp a source image I_s representing a rendered view of a scene to a target image I_w that resembles the scene from a different viewpoint in space or time. In the next section we describe the concept of the warp in detail and then introduce an efficient solution method which employs iterative methods to solve the warp efficiently.

3.1. Image Warp Definition

We define the warp as a vector field $V : \mathbb{R}^2 \to \mathbb{R}^2$ that describes how each point in the source image should be translated in order to produce I_w. For a particular point $\mathbf{x}_s$ in the source image, the warped image coordinates $\mathbf{x}_w$ are given by

$$\mathbf{x}_w = \mathbf{x}_s + V(\mathbf{x}_s), \tag{1}$$

with $\mathbf{x}_s, \mathbf{x}_w \in \mathbb{R}^2$. For a particular pixel at point $\mathbf{x}_w$ in the warped image I_w, we wish to find the location(s) $\mathbf{x}_s$ in the source image I_s that satisfy Equation 1. When multiple solutions $\mathbf{x}_s$ exist we take the one with the minimum depth $z(\mathbf{x}_s)$ to maintain depth order. We now introduce the specific warp fields studied in this work.

Temporal Reprojection The warp field for temporal reprojection is given by per-pixel motion vectors, a common by-product of modern rendering pipelines which indicate the screen space motion of the rendered surfaces [Ros07]. These motion vectors can be computed by projecting the position of the scene geometry at two points in time and computing the derivative using finite differences. The warp field is then defined as

$$V_{temporal}(\mathbf{x}_s, t) = tM(\mathbf{x}_s) \tag{2}$$

where M is the motion vector field, and t is the difference in time to reproject the viewpoint. From second order finite differences we can obtain acceleration vectors A and can perform nonlinear temporal reprojection using the Taylor expansion

$$V_{nonlinear}(\mathbf{x}_s, t) = tM(\mathbf{x}_s) + \frac{t^2}{2}A(\mathbf{x}_s). \tag{3}$$

We will use temporal reprojection to increase rendering frequency and simulate motion blur.

Spatial Reprojection Spatial reprojection corresponds to moving the viewpoints position in the scene. A shift orthogonal to the view direction can be achieved by shifting the surfaces in the image in the opposite direction. We define the shift in view space which can be projected into screen space as follows:

$$V_{spatial}(\mathbf{x}_s, u, v) = \left(\frac{-u\lambda_u}{z(\mathbf{x}_s)}, \frac{-v\lambda_v}{z(\mathbf{x}_s)}\right) \tag{4}$$

where (u, v) are the horizontal and vertical components of the viewpoint shift, λ_u and λ_v are the first two constants from the projection matrix diagonal and z is the rendered depth buffer. We use spatial reprojection to efficiently render stereoscopic 3D content and to simulate defocus blur.

General Warp Fields We emphasize that we have not placed restrictions or requirements on the form of V (only that it can be evaluated at run-time). The method we will present can easily be applied to other warp fields (including combinations of the presented warp fields) or to more general image warping scenarios.

3.2. Fixed Point Iteration

In this work we introduce the application of iterative methods to this problem, in particular *fixed point iteration* (FPI). We will show that for our purposes FPI generally converges quickly and behaves in a well defined and deterministic way. In this section we describe the basic modes of behavior of the iteration before describing its application to image warping in the following sections.

Fixed Point Iteration Definition For convenience we define a new function $G : \mathbb{R}^2 \to \mathbb{R}^2$ as

$$G(\mathbf{x}_s) = \mathbf{x}_w - V(\mathbf{x}_s) \tag{5}$$

and rewrite Equation 1

$$\mathbf{x}_s = G(\mathbf{x}_s). \tag{6}$$

The value $\mathbf{x}_s = \mathbf{x}^*$ that satisfies Equation 6 corresponds to a *fixed point* of G; the result of G evaluated on $\mathbf{x}^*$ is $\mathbf{x}^*$. FPI solves equations of this form by generating a set of iteration points (*iterates*) $\mathbf{x}_i$ using the recurrence relation:

$$\mathbf{x}_{i+1} = G(\mathbf{x}_i). \tag{7}$$

Seeded with an initial value $\mathbf{x}_0$, the method computes successive iterates $\mathbf{x}_i$ through repeated applications of G. In the next section we analyze the behavior of FPI and introduce the conditions necessary to ensure robust and reliable convergence.

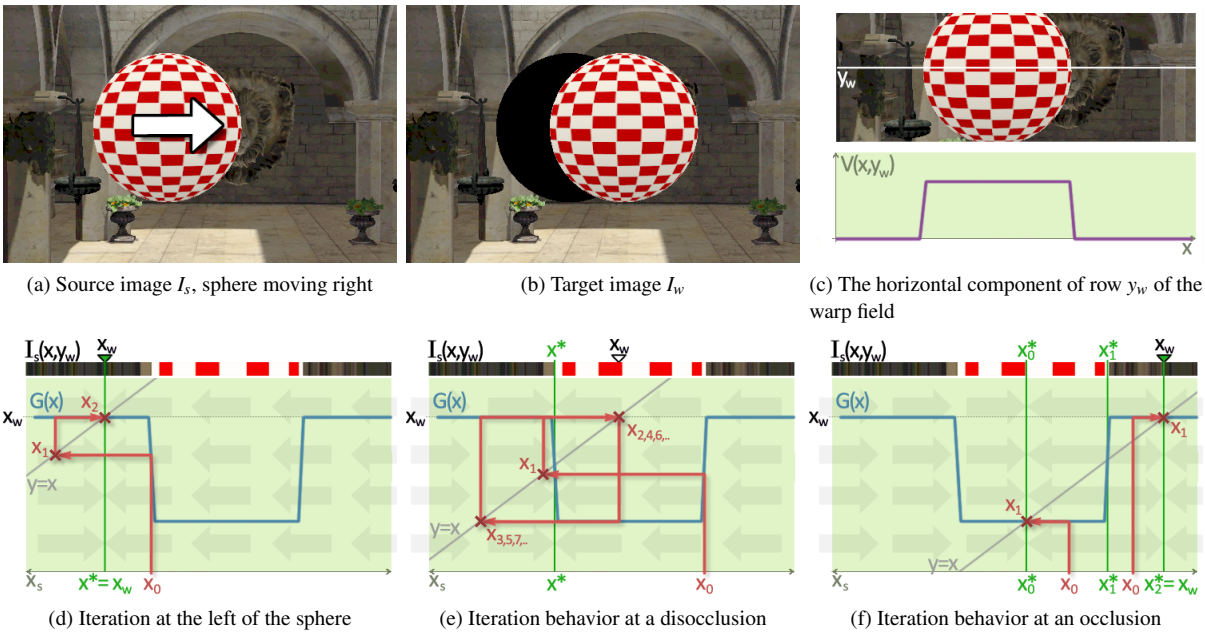
Iteration Behavior We decompose the behavior into three cases using the example scene depicted in Figure 1, in which a sphere is moving horizontally against a stationary background. We reduce our focus to one dimension by considering a *single horizontal slice of the motion vectors taken across the sphere at* $y = y_w$ and considering only the horizontal component of the warp field V (plotted in Figure 1c).

The iteration behavior varies depending on the target pixel location x_w. Figures 1d, 1e and 1f plot $G(x)$ (Equation 5) at three different locations x_w. To help provide orientation the *source image* intensities from the slice $y = y_w$ are shown above each chart. The solution points are labeled x_i^* and lie at the intersection between the line $y = x$ and $G(x)$, i.e. at the fixed points of G. The trajectories of the iteration are shown using *cobweb plots*, where the iterates are labeled x_i and the process of evaluating G on the current iterate x_i to yield the next iterate x_{i+1} (Equation 7) is visually represented by the iteration arrows moving vertically to touch the curve G (representing an evaluation of $G(x_i)$) and then moving horizontally to the line $y = x$ (representing the assignment of $G(x_i)$ to the next iterate x_{i+1}). A useful rule to note is that *the iteration will move to the right if G is above* $y = x$ *at the current iterate, otherwise it will move to the left*. This is indicated on the background of the charts.

Unique solution In Figure 1d the target pixel at x_w lies on the left hand side of the sphere in the source image on the stationary background. Since there is only one intersection between G and the line $y = x$ in Figure 1d, the iteration is attracted uniformly towards the solution x^* (as indicated by the attraction arrows on the background of the chart) and will converge to x^* regardless of the starting point x_0.

No solutions In Figure 1e x_w lies on the sphere in the source image. The particular choice of iteration start point x_0 places the next iterate x_1 in close proximity to the solution x^*. However the large slope of G around x^* pushes the iteration away into a surrounding orbit. This steep slope corresponds to interpolation across the discontinuity in motion at the left-hand edge of the sphere at which a disocclusion occurs (x^* is also labeled on the source image intensities above the chart), and no solution exists to the image warp. In Figure 1b disoccluded pixels are shaded black. In such cases this orbital trajectory is always observed, and we will exploit this behavior in the next section to fill disocclusions.

Multiple solutions The target pixel x_w in Figure 1f lies on the background on the right hand side of the sphere and there are three solution points x_0^* to x_2^*, which lie on the sphere, on the discontinuity at its right hand edge, and on the background respectively. Applying the general rule regarding the iteration behavior around the solution at x_1^* reveals interesting behavior. On the left side of x_1^*, G is below the line $y = x$ and the iteration will move to the left in this region, *away* from x_1^*. On the right side of x_1^* the iteration is also repelled in a similar manner. Thus x_1^* repels the iteration towards surrounding solutions as shown by the two iteration trajectories on either side of x_1^*. The target pixel lies on an overlapping region and *the solution obtained from the iteration depends*



(a) Source image I_s, sphere moving right

(b) Target image I_w

(c) The horizontal component of row y_w of the warp field

(d) Iteration at the left of the sphere

(e) Iteration behavior at a disocclusion

(f) Iteration behavior at an occlusion

Figure 1: An illustration of the behavior modalities of fixed point iteration. These examples are analyzed in Section 3.2.

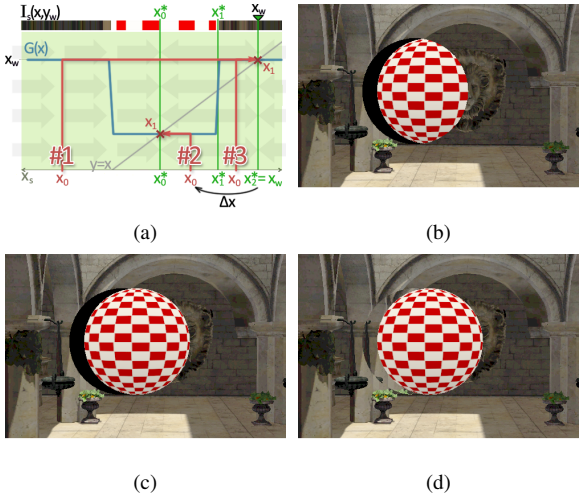
on the iteration start point x_0. In the next section we apply FPI to image warping.

3.3. Iterative Warping with Heuristics

As discussed in the previous section and illustrated in Figures 1d-1f, if there is a single solution to the warp the iteration will converge regardless of the start point. If no solution exists, the iteration will not converge but will orbit the disocclusion. If there are multiple solutions, the outcome depends on the iteration start point.

If the iteration is simply started from the target pixel location ($\mathbf{x}_0 \leftarrow \mathbf{x}_w$), it will resemble trajectory #3 on Figure 2a and converge to the background, failing to resolve the overlapping surface (Figure 2b). A simple heuristic for this specific case is to offset the iteration δx pixels to the left of $\mathbf{x}_w$, past $\mathbf{x}_1^*$ and onto the sphere, in which case the overlapping surface is reconstructed correctly (Figure 2c).

We applied this heuristic to stereoscopic reprojection, in which the camera is shifted some units horizontally to create an offset view. Figure 2 closely resembles a spatial reprojection of the camera to the left, and the principles are the same. The size of the offset depends on the maximum width of the overlap which for this application is a function of the stereoscopic parameters and the relative depths of objects. We found a simple fixed offset worked well in practice (see Figure 6). We also applied this heuristic variant to temporal reprojection to upsample the racing game from 30Hz to 60Hz (c.f. the video material). In this game the screen space



(a)

(b)

(c)

(d)

Figure 2: Heuristics-based reprojection. (a) A cobweb plot of some example trajectories at the right hand side of the sphere. (b) When the current pixel coordinates are used as the iteration start point, the behavior resembles trajectory #3 with convergence to solution x_2^* on the background. (c) The iteration start point is offset δx pixels to the left, starting the iteration on the sphere and yielding the desired solution x_0^*. (d) Inpainting using surrounding texture.

motion of surfaces is usually relatively uniform, with some variation in relative speeds due to motion parallax. We therefore start the iteration an additional frame up-stream in the visual flow (determined by the source frame motion vector at the target pixel location) which correctly locates many of the overlapping surfaces. Yang et al. [YTS*11] report additional heuristics for the case where multiple input frames are available for reprojection.

In-Painting The tendency of the iteration to orbit disocclusions (Figure 1e) results in rudimentary in-painting behavior. The iteration alternates between the foreground and background surfaces, and the background solution can be selected by depth comparison. The result will be a clone of the surrounding background texture in the disocclusion. For different target pixel coordinates $\mathbf{x}_w$ in the warped view G is translated vertically accordingly, the orbit slides smoothly along the line $y = x$.

How well this in-painting strategy works in practice depends on a number of factors including the size of the disocclusion, the visual saliency of the affected region and the background texture. Figure 2d illustrates a severe case in which the disocclusion is large and the background texture contains directional features that are broken at the edge of the disocclusion. Despite these issues similar hole filling strategies have already been applied in production scenarios [SKS11, HHE11]. Additionally we emphasize that when such a strategy is not suitable, other approaches can be used such as re-rendering disoccluded regions of the scene (we discuss such strategies in detail in Section 6).

3.4. Robust Iterative Warping

Instead of selecting the iteration start point using heuristics, we introduce results that guarantee FPI convergence under certain conditions, and lead to our robust iterative image warping algorithm.

Fixed Point Theorem The *fixed point theorem* states that, provided that the magnitude of the slope of G is less than unity in some convex region R around a solution x^*, and provided that the iteration starting point is chosen within R, then FPI is *guaranteed* to converge to x^*.

This restriction on the slope of G ensures that it does not cross the line $y = x$ from below as for x_1^* in Figure 1f, and that where it does cross $y = x$ from above it does so in a well behaved way unlike the sharp drop in Figure 1e. Note that this is not a *necessary* condition for convergence, for example in Figure 1d there is only a single intersection between G and $y = x$ at x^* and the iteration is uniformly attracted towards x^*, regardless of the slope of G in the surrounding area.

We denote the slope of G as the *convergence term* ρ. In Appendix A we provide a formal definition for the slope and generalize it to two dimensions, obtaining the following:

$$\rho = \max_i |\lambda_i| < 1, \tag{8}$$

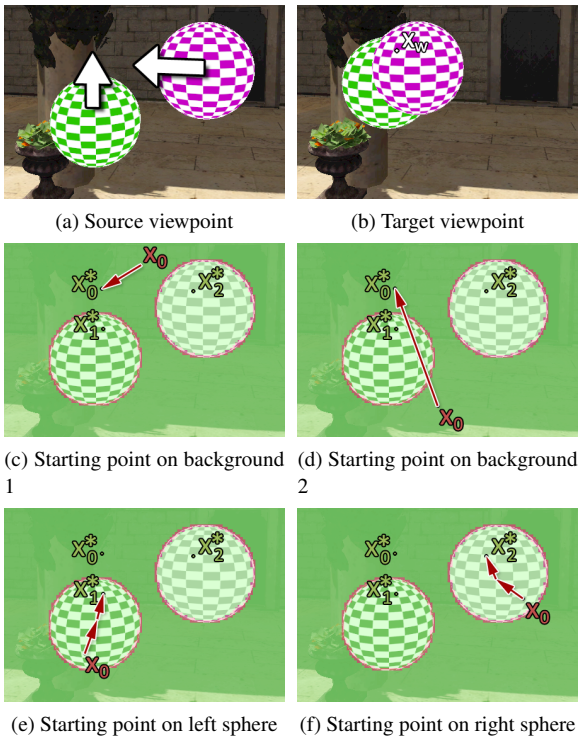
where λ_i are the eigenvalues of the Jacobian of G (the 2×2 matrix of its partial derivatives). In the remaining sections we will refer to $\rho < 1$ as the *convergence condition*.

Figure 3 illustrates the practical implications of the convergence condition for a scene in which two spheres are moving against a stationary background from separate locations in the source image Figure 3a to an overlapping configuration in the warped image Figure 3b. Due to the overlap there are multiple solution points $\mathbf{x}_0^*$, $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$ labeled on Figures 3c-3f. The pixels in Figures 3c-3f are shaded green if the convergence condition is satisfied and red otherwise. The discontinuities in the warp field (red) divide the source image into a number of smooth regions (green). By the fixed point theorem, if there is a solution $\mathbf{x}^*$ in such a smooth region, and the first iterate $\mathbf{x}_0$ is placed inside the region, convergence to $\mathbf{x}^*$ is guaranteed. Figures 3c-3f provide examples of this behavior.

Sampling Strategy Although an advanced sampling strategy such as cubic spline interpolation may produce more accurate results for some warp types [CLC*08] we restrict our focus in this work to bilinear interpolation for simplicity.

Interpolation across discontinuities such as geometry edges in the scene may introduce artificial solutions (c.f. solutions $\mathbf{x}^*$ and $\mathbf{x}_1^*$ in Figures 1e and 1f respectively). Although identifying such discontinuities in a discretely sampled signal is inherently ambiguous, Mark et al. [MMB97] discuss a practical approach that uses geometric information such as depths and normals. We opt for a general measure that does not depend on such geometric information, namely the convergence condition (Equation 8). If this condition is not satisfied between 4 pixels, FPI will not converge to a solution there, and we sample the texture using nearest neighbor filtering. Figure 4 illustrates the resulting reconstruction in one dimension.

Robust Warp Refer to Figure 5 for an overview of the robust warp, which is conceptually similar to the adaptive quad tree compressed warp of Chen and Williams [CW93], augmented with FPI to compute an accurate solution. The convergence condition provided by the fixed point theorem (Equation 8) is used as the split criteria for the quad regions. Ensuring that the convergence condition is satisfied over source image pixels represented by each leaf node avoids the situation in Figure 1f in which the solution depends on the iteration start point. We then rasterize the leaf nodes at their warped locations, interpolating the starting point x_0 from the source image locations of the quad vertices, and then iterate to compute the solution. If the iteration does not converge, the fragment is discarded. Otherwise the source image and depth are sampled at the converged location and both are

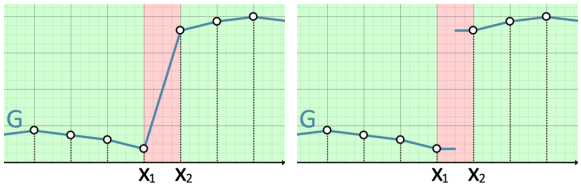


(a) Source viewpoint (b) Target viewpoint

(c) Starting point on background 1 (d) Starting point on background 2

(e) Starting point on left sphere (f) Starting point on right sphere

Figure 3: Iteration trajectories (red arrows) for a selection of start points $\mathbf{x}_0$ in an example scene in which (a) is warped forward in time to (b). Each solution x_i^* indicates a point on a surface in the source view (a) which warps to the target pixel at $\mathbf{x}_w$ in (b). The divergent pixels (shaded red) form boundaries around convergent regions (shaded green), within which convergence to any solution is guaranteed.



(a) (b)

Figure 4: Adaptive sampling in the presence of a steep gradient/discontinuity (shaded red). Using naive interpolation results in the reconstruction in (a). The convergence term (equivalent to $|G'(x)|$ in 1D) is greater than 1 between x_1 and x_2, violating the convergence condition (Equation 8), and FPI will not converge to any solution in this region. Switching to point sampling (b) extrapolates the signal into the discontinuity.

(a) (b)

Figure 5: Robust iterative warp. The source image is recursively subdivided until no quad straddles a discontiuity in the warp field (a). To perform the reprojection, the quads are rasterized at their warped locations (b). The iteration starting point $\mathbf{x}_0$ is then interpolated from the quad and used as the starting point of the iteration.The iteration (red arrow in (b)) is guaranteed to converge to the solution $\mathbf{x}^*$.

written to the render target. Writing depth allows the hardware depth buffer to efficiently maintain correct visibility in the presence of overlaps.

Adaptive Subdivision The source image is tiled with a set of initially coarse quads, which are then recursively subdivided based on the convergence condition $\rho < 1$ until no quad straddles a discontinuity in the warp field. We implement this process as a geometry shader in a similar manner to Didyk et al. [DRE*10]. To support fast queries of the convergence term ρ over areas of pixels, a number of max-reductions are performed over ρ as a pre-process. Although ideally the convergence condition is computed only across the pixels interior to a region, it is non-trivial to perform the max-reductions of ρ in this irregular way and we settle for at the occasional over-subdivision that may result if discontinuities (i.e. large values of ρ) lie along borders between regions.

Quad Warping To avoid thin cracks appearing along quad boundaries in the warped view, the warped quads need to be large enough to completely bound the warped positions of the pixels. One way to compute such a bound would be to, in addition to performing a max-reduction on the convergence terms ρ in the pre-processing stage, compute a hierarchy of bounding boxes for the warped positions of the underlying pixels, and use these bounding boxes as the proxy geometry primitive. However, we found that computing the warped position of each vertex of the quad and using the bounding box of these positions (expanded by half a pixel to account for the pixel coverage) as the warped quad worked well in practice. The sampling strategy discussed in Section 3.4 and illustrated in Figure 4 is used to avoid interpolation across discontinuities, which would otherwise result in large stretched quads in screen space.

4. Implementation

The heuristic variant of our approach, used to render the racing game results, consists of a single fixed point iteration pixel shader executed over a full screen quad. The stereoscopic reprojection result (Figure 6) applies a simple fixed offset as described in Section 2. The temporal reprojection result (accompanying video material) reprojects each 30Hz frame 16.67ms forward in time to generate an intermediate 60Hz frame. In both cases we fix the number of iterations to 3, remove the convergence thresholding and unroll the iteration loop, yielding a minimal and high performance reprojection kernel.

For the robust variant, the generality of our algorithm allows us to implement a single image warping kernel, decoupled from the definition of the warp field. The warp field we use is a linear combination of the warps introduced in Section 3.1, including the nonlinear motion vectors. No advanced techniques such as optimizing the shader code by hand have been employed, and we therefore expect there is room for further performance improvements.

Although we have presented the quad subdivision and warping stages paired together, for multiple reprojections the quad subdivision stage can be factored out as a preprocess to generate the geometry used for all reprojections. To demonstrate this we accumulate many jittered views to simulate motion and defocus blur [HA90]. We sample the parameter values t, u and v from a Poisson-ball distribution, and use a simple box filter when accumulating the views. To ensure the quads are subdivided to a sufficiently fine level for all reprojections, we take the maximum convergent term over the ranges of the reprojection parameters:

$$\rho_{max} = \max_{t,u,v} \rho. \tag{9}$$

Pseudocode for our accumulation buffer implementation is provided in Algorithm 1. The reprojection is computed in to a temporary render target and accumulated in a separate pass due to incompatible GPU states (accumulation requires blending to be enabled, while the reprojection requires that blending is disabled in order for overlaps to be handled correctly).

Algorithm 1 Complete Reprojection Pipeline

1: Obtain input data (colors, depths, motion vectors, ...)
2: Compute per-pixel converge terms ρ
3: Perform max-reductions to build convergence term hierarchy
4: Subdivide quad geometry
5: **for each** reprojected view parameters (t,u,v) **do**
6: Clear temporary render target and depth buffer
7: Bind FPI fragment program to GPU
8: Warp and rasterize quad geometry
9: Accumulate result onto final render target
10: **end for**

Figure 6: Stereoscopic reprojection with offset heuristics (Section 3.3) fitted as a post process to an existing racing game. The overhead per view is only 0.85ms on Sony PlayStation 3 hardware. For anaglyph both views can be computed in a single pass in under 1ms.

5. Results and Analysis

Performance All of the racing game examples (e.g. Figure 6) use the heuristic variant as mentioned. On the PlayStation 3 NVIDIA RSX GPU a single reprojection pass takes 0.85ms on average at 720p (1280×720) pixel resolution. This is a strong result for this hardware.

The other results were rendered on an NVIDIA GeForce GTX580 GPU, also at 720p resolution. On this hardware the heuristic reprojection takes under 0.2ms on average. Refer to Table 1 for timing results averaged over 64 reprojection passes. To apply load to the hardware we subdivided the scene to increase the polygon count to around 5 million and added Perlin noise evaluations to the fragment shaders, resulting in a forward render time of 15.29ms, approximately the frame budget of a 60Hz application. For RRC (Nehab et al.) we timed *the cache lookup pass only* which produces a result with disocclusions equivalent to the IBR methods. This pass takes around 6.64ms which is significantly less than the forward render but still relatively expensive due to the geometry processing cost, which highlights the disadvantage of the coupling to the scene's geometry. The warping method from Mark et al. is decoupled from the scene description and executes in around 2.5ms. This method is also bound by geometry processing due to the per-pixel primitives (approximately 1.8 million triangles at 720p), in contrast to our adaptive subdivision which usually generates around 60k triangles. For our method a single reprojection pass takes just 0.7ms, a 3.5× speedup over the warp from Mark et al.

Error Analysis In Figure 8 we quantify the error in the result from each approach for a scene in which the camera is reprojected both spatially (horizontally and vertically by 0.5m) and temporally (forward by 32ms). CIE94 error is computed under the standard illuminant D65. The first set of error comparisons compare the methods for a non-planar

Figure 7: A dynamic gorilla scene with accumulation buffering.

	Ref.	Nehab 2007	Mark 1997	Ours
Quad Subdiv.	–	–	–	0.48
Reprojection	15.29	6.64	2.49	0.70

Table 1: Performance in milliseconds of different reprojection strategies for the Sponza scene.

surface (the lion head protrudes from the wall). The methods produce nearly identical results, with the same mean error to two significant digits. The second set examines the sphere which is shifted from its location in the source image. The sharp contrasts in the checkered texture highlight the blurring caused from the bilinear filtering used to sample the intensities from the source image. As Mark et al. note, their method tends to omit pixels at object boundaries as triangles that straddle discontinuities in the warp are discarded. For RRC the same problem occurs if the depths from the cache are linearly interpolated, which leads to incorrect depth readings near boundaries. As Nehab et al. note this issue can be addressed for smooth surfaces by point sampling the cached depths which leads to the relatively complete reconstruction shown. Using our adaptive sampling strategy (Figure 4) our method recovers most of the edge pixels. In general we expect small inconsistencies at edges because the exact locations of geometry edges are lost when the source image is rendered, whereas RRC rasterizes the scene during reprojection, obtaining the true geometry edges.

As further analysis we illustrate the convergence of the iteration in Figure 9 and compare linear and nonlinear motion blur in Figure 10.

6. Discussion and Future Work

In this paper, we have introduced a novel robust image warp based on iterative methods. We implemented our method into a high performance console game that renders frames in approximately 32ms to synthesize convincing new frames in less than 1ms, which demonstrates the remarkable

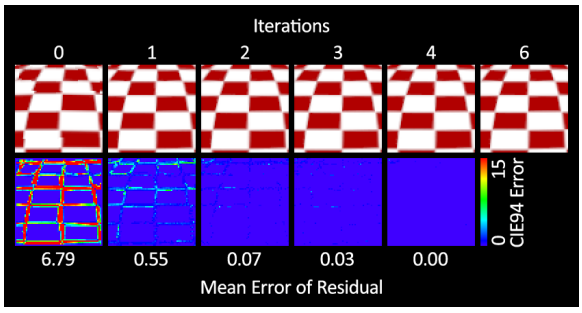


Figure 9: The result and residual error after each iteration for a curved patch on the sphere.

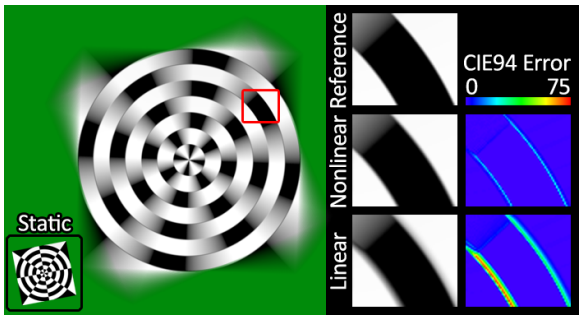


Figure 10: Our method is able to handle non-linear warps, such as the non-linear motion blur rendered here by accumulating 100 reprojected images.

speed gains achievable by exploiting coherency in rendering pipelines.

In such high performance rendering scenarios the scene layers are flattened onto a single opaque visible surface layer, which can lead to error in reprojections. Although such error is reduced in practice when small reprojection offsets are used [NSL*07, SKS11], with improved algorithms and hardware it may become feasible to render the scene into layers composited in post (as in off-line rendering) which retains translucent layers and eliminates disocclusions [YHGT10], or to perform the reprojection from multiple input images such as in two frame interpolation to reduce the number of disoccluded pixels. This could be performed in a single pass using the heuristics variant of our approach as demonstrated in the context of temporal reprojection by Yang et al. [YTS*11], or by using our robust algorithm to reproject multiple input views onto a single target view (allowing the depth buffer to enforce depth order).

Although our warping algorithm could produce anti-aliased images by further subdividing the warp quads to the level of sub-pixel samples, in high definition real-time applications it is usually not an option to use the render output at this stage because it is subsequently resolved to display pixel resolution and post-processed with expensive

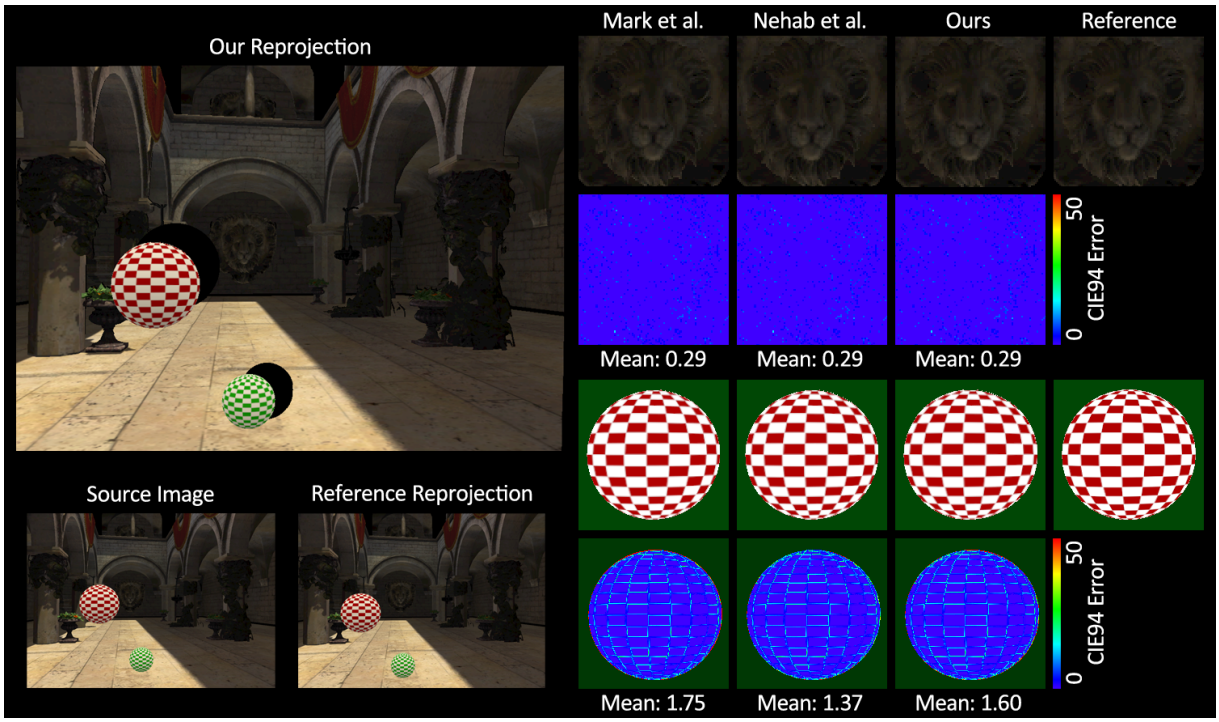


Figure 8: Our reprojection along with the source image and reference solution are shown on the left hand side. The right hand side compares our result with existing methods for a non-planar background region and for a highly discontinuous region.

shaders, which would have to be repeated for each reprojection. This is the case for the racing game example. While we did not observe objectionable aliasing, another direction for future work would be to integrate our reprojection with one of the image-based anti-aliasing approaches (popularized by Reshetov [Res09]) which could be efficiently applied to the image after the reprojection.

Other iterative methods such as Newton Raphson Iteration (NRI) may also be applicable to image warping. Although NRI generally has a faster convergence rate, it may not always converge and generally requires supervision. Future work use NRI to augment or completely replace FPI.

Our method may be applicable to other contexts such as latency compensation, i.e. to synthesize new frames based on user input to provide immediate feedback. This may be useful for head-mounted displays where low latency is critical or for cloud gaming services such as *OnLive*. Another potential application is the synthesis of input views to feed next generation multiscopic displays which require many input views, making transmission impractical.

References

[And10] ANDREEV D.: Real-time frame rate up-conversion for video games: or how to get from 30 to 60 fps for "free". In *ACM SIGGRAPH 2010 Talks* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 16:1–16:1. 2

[Bow10] BOWLES H.: *Efficient Real-Time Stereoscopic 3D Rendering*. Master's thesis, ETH Zurich, 2010. 2

[CLC*08] CHEN M., LU W., CHEN Q., RUCHALA K. J., OLIVERA G. H.: A simple fixed-point approach to invert a deformation field. *Medical Physics 35*, 1 (2008), 81–88. 2, 5

[CW93] CHEN S. E., WILLIAMS L.: View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 279–288. 2, 5

[DER*10] DIDYK P., EISEMANN E., RITSCHEL T., MYSZKOWSKI K., SEIDEL H.-P.: Perceptually-motivated real-time temporal upsampling of 3D content for high-refresh-rate displays. *Computer Graphics Forum (Proceedings Eurographics 2010, Norrköpping, Sweden) 29*, 2 (2010), 713–722. 2

[DRE*10] DIDYK P., RITSCHEL T., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Adaptive image-space stereo view synthesis. In *Vision, Modeling and Visualization Workshop* (Siegen, Germany, 2010), pp. 299–306. 2, 6

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 43–54. 2

[HA90] HAEBERLI P., AKELEY K.: The accumulation buffer: hardware support for high-quality rendering. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 309–318. 7

[HHE11] HALL C., HALL R., EDWARDS D.: Rendering in cars

2. In *ACM SIGGRAPH 2011 courses* (New York, NY, USA, 2011), SIGGRAPH '11, ACM. 5

[KTI*01] KANEKO T., TAKAHEI T., INAMI M., KAWAKAMI N., YANAGIDA Y., MAEDA T., TACHI S.: Detailed shape representation with parallax mapping. In *In Proceedings of the ICAT 2001* (2001), pp. 205–208. 2

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 31–42. 2

[Mar98] MARCATO R.: *Optimizing an Inverse Warper*. Master's thesis, Massachusetts Institute of Technology, 1998. 2

[Max96] MAX N.: Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers. In *Eurographics Rendering Workshop* (New York City, NY, 1996), Eurographics, Springer Wein, pp. 165–174. 2

[McM97] MCMILLAN L.: *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina (Chapel Hill), 1997. 2

[MMB97] MARK W. R., MCMILLAN L., BISHOP G.: Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics* (New York, NY, USA, 1997), I3D '97, ACM, pp. 7–ff. 2, 5

[NSL*07] NEHAB D., SANDER P., LAWRENCE J., TATARCHUK N., ISIDORO J.: Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2007), Eurographics Association, p. 35. 1, 8

[Res09] RESHETOV A.: Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, New York, USA, 2009), ACM, pp. 109–116. 9

[Ros07] ROSADO G.: Motion blur as a post-processing effect. *GPU Gems 3* (2007), 575–582. 2

[SaLY*08] SITTHI-AMORN P., LAWRENCE J., YANG L., SANDER P., NEHAB D.: An improved shading cache for modern GPUs. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2008), Eurographics Association, pp. 95–101. 1

[SGHS98] SHADE J., GORTLER S., HE L. W., SZELISKI R.: Layered depth images. In *Proceedings of SIGGRAPH 98* (1998), ACM New York, NY, USA, pp. 231–242. 2

[SKS11] SOUSA T., KASYAN N., SCHULZ N.: Secrets of cryengine 3 graphics technology. In *ACM SIGGRAPH 2011 courses* (New York, NY, USA, 2011), SIGGRAPH '11, ACM. 2, 5, 8

[SLS*96] SHADE J., LISCHINSKI D., SALESIN D., DEROSE T., SNYDER J.: Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In *SIGGRAPH 96 Conference Proceedings* (1996), Addison Wesley, pp. 75–82. 2

[SS96] SCHAUFLER G., STURZLINGER W.: A Three-Dimensional Image Cache for Virtual Reality. In *Proceedings of Eurographics* (1996), Eurographics Association, pp. 227–236. 2

[TK96] TORBORG J., KAJIYA J. T.: Talisman: commodity real-time 3d graphics for the pc. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 353–363. 2

[WDG02] WALTER B., DRETTAKIS G., GREENBERG D.: Enhancing and optimizing the render cache. In *Proceedings of the 13th Eurographics Workshop on Rendering* (June 2002), vol. 10, ACM Press. 2

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive Rendering using the Render Cache. In *Proceedings of the 10th Eurographics Workshop on Rendering* (June 1999), vol. 10, Springer-Verlag/Wien, pp. 235–246. 2

[Wes90] WESTOVER L.: Footprint evaluation for volume rendering. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 367–376. 2

[YHGT10] YANG J. C., HENSLEY J., GRUEN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. pp. 1297–1304. 8

[YTS*11] YANG L., TSE Y.-C., SANDER P. V., LAWRENCE J., NEHAB D., HOPPE H., WILKINS C. L.: Image-based bidirectional scene reprojection. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2011) 30*, 6 (2011). 2, 5, 8

[ZPvBG02] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics 8* (July 2002), 223–238. 2

Appendix A: Convergence Condition

The formal definition of the convergence condition is based on *Lipschitz continuity* which requires that the slope of all secant lines to G be bounded by a *Lipschitz constant* K, i.e.

$$\frac{d(G(\mathbf{x}_1), G(\mathbf{x}_2))}{d(\mathbf{x}_1, \mathbf{x}_2)} \leq K, \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in R \subseteq \mathbb{R}^n, \tag{10}$$

where R is some region and d is a distance metric. If G is Lipschitz continuous with $K < 1$ the *fixed point theorem* guarantees that if there is a solution $\mathbf{x}^*$ in R then the iteration will converge monotonically to $\mathbf{x}^*$ provided that the iteration start point $\mathbf{x}_0$ lies in R. A practical way to check that the convergence condition Equation 10 is satisfied over some region R is to check whether $||G'(\mathbf{x})|| < 1, \forall \mathbf{x} \in R$ where G' is the Jacobian of G (which we compute using finite differences). This is true for all induced norms $||\cdot||$. The infimum (minimum possible value) of all such norms is given by the *spectral radius* ρ defined as

$$\rho(G') = \max_i(|\lambda_i|), \tag{11}$$

where λ_i are the eigenvalues of G'. This provides an optimal norm-independent measure of convergence. The final *convergence condition* is

$$\rho(G'(\mathbf{x})) < 1, \forall \mathbf{x} \in R. \tag{12}$$

Acknowledgements For their stimulating conversations and support we thank Peter-Pike Sloan and those who worked at Black Rock Studio, including but not limited to Romain Pacanowski, Yuriy Yemelyanov, George Parrish, Lei Yang, Joe Palmer, Jim Cox, Nathan Gouveia and Tom Williams. Sponza data set courtesy of Mark Dabrovic and Crytek GmbH. Figures 6, 7 © Disney Enterprises, Inc.