Iterative Depth Warping

SUNGKIL LEE and YOUNGUK KIM, Sungkyunkwan University, South Korea
ELMAR EISEMANN, Delft University of Technology, Netherlands

Fig. 1. Comparison of our depth warping (c) and the reference depth (d) with their difference (e). Given depth/motion buffers (b) at a known view (a), our image-based warping solution can generate a high-quality depth buffer (here, SSIM=0.998; PSNR 32 dB) without costly geometric rendering at a novel view.

This article presents an iterative backward-warping technique and its applications. It predictively synthesizes depth buffers for novel views. Our solution is based on a fixed-point iteration that converges quickly in practice. Unlike the previous techniques, our solution is a pure backward warping without using bidirectional sources. To efficiently seed the iterative process, we also propose a tight bounding method for motion vectors. Non-convergent depth holes are inpainted via deep depth buffers. Our solution works well with arbitrarily distributed motion vectors under moderate motions. Many scenarios can benefit from our depth warping. As an application, we propose a highly scalable image-based occlusion-culling technique, achieving a significant speedup compared to the state of the art. We also demonstrate the benefit of our solution in multi-view soft-shadow generation.

CCS Concepts: • **Computing methodologies** → **Rasterization**; *Visibility*;

Additional Key Words and Phrases: Depth warping, GPU rendering, occlusion culling, soft shadows

ACM Reference Format:
Sungkil Lee, Younguk Kim, and Elmar Eisemann. 2018. Iterative Depth Warping. *ACM Trans. Graph.* 37, 1, Article 1 (January 2018), 13 pages. https://doi.org/10.1145/3190859

1 INTRODUCTION

The depth buffer is a fundamental basis for diverse computer graphics techniques. Modern graphics-processing-unit (GPU) pipelines generate the depth buffer along with the associated color buffer in the frame buffer, and use it to test the visibility of incoming fragments. For many modern techniques in recent rendering pipelines, the depth buffer also plays an important role in rendering the final image. Common examples include shadow mapping, global illumination, occlusion culling, ambient occlusion, and many others. It would be beneficial to have access to it prior to the actual rendering and at lower cost.

Typically, the depth buffer is created in a full rendering pass, which is usually costly. Fortunately, in many scenarios, the previous depth buffers are readily available, especially in deferred-rendering pipelines. This observation led us to explore an efficient depth-buffer generation method. Depending on scene complexity, our solution is much faster than regular rendering, thereby reducing the overall costs of many applications.

Our solution builds upon *warping*, which has a long history, e.g., in the context of view interpolation. Warping is a fast and simple technique that matches real-time requirements, in contrast to many other advanced techniques in image processing and computer vision. Nonetheless, warping was previously mostly successful for color images, where approximation errors, if only weakly perceivable, are often acceptable. In contrast, small errors in the depth buffer can have an important impact on visibility. Hence, the warping needs to be precise or conservative, making an efficient depth-buffer prediction very challenging.

Successful warping solutions exist mostly for constrained scenarios (e.g., stereoscopic view synthesis as 1-D warping). For general object/camera motions, fewer approaches exist. The warping is typically realized by *forward* or *backward* mapping. The forward mapping displaces a source pixel to a destination pixel with its motion vector, and the backward mapping gathers source pixels whose motions can displace them to the destination pixel. Forward mapping techniques map pixels to splats/sprites [Zwicker et al. 2002] or polygonal meshes [Mark et al. 1997]. However, this step can involve mapping millions of vertices for typical display resolutions. Adaptive subdivision of the full-screen quad [Didyk et al. 2010a,b] helps, but is still costly for high depth complexity. Backward mapping gathers source pixels, and can be much more efficient for higher resolutions, as it avoids heavy vertex processing or atomic depth writing. It is difficult to find an analytic solution for the gathering, and heuristics [Andreev 2010] or manual search [McMillan Jr

Authors' addresses: Sungkil Lee; Younguk Kim, Sungkyunkwan University, 2066, Seoburo, Jangan-gu, Suwon, 16419, South Korea; Elmar Eisemann, Delft University of Technology, Van Mourik Broekmanweg 6, Delft, 2628CD, Netherlands.

0730-0301/2018/1-ART1 $$15.00
https://doi.org/10.1145/3190859

1997] are typically employed. Recently, as an alternative, the *fixed-point* iteration drastically relaxed the problem due to its fast convergence [Bowles et al. 2012; Yang et al. 2011]. In particular, the work of Bowles et al. [2012] forms a great basis for efficient warping.

We present an efficient and scalable depth-buffer warping technique for predictive depth-buffer generation (Fig. 1). Our depth warping improves the previous color-image warping technique based on the fixed-point iteration [Bowles et al. 2012], which was originally intended to synthesize a novel view from the previous frame's color and depth buffers. Instead, we adapt the approach for depth-buffer warping; we use the depth buffer as an input to our warping. Unlike color-image warping, warping depth values requires efficient yet more careful inpainting of potential holes that cannot be filled via warping. We present a high-quality depth-inpainting technique based on deep depth buffers, which is useful for multi-view syntheses. We demonstrate the benefit of our technique via example applications.

Our work differs from previous approaches, as we address pure *backward-only* warping, while the majority of previous solutions have relied on forward mapping. Similarly, Bowles et al. [2012], which motivated our work, partly uses forward mapping to initialize the warping. Additionally, most previous work used warping for view interpolation/morphing. Thus, the warped view is an output, while our depth warp is an input to subsequent processing.

Applications of our depth warping relate to previous temporal-coherence techniques [Scherzer et al. 2012]. Nonetheless, they hardly addressed the reduction of geometric processing; typically, they reduced the overhead of fragment processing (e.g., amortized sampling). Ours differs in this respect, which adds to the novelty.

Our first application strongly utilizes our depth-buffer warping, to enable a scalable low-latency hierarchical occlusion-culling technique, relying on a conservative depth inpainting and efficient depth warping from the previous frame. We avoid an explicit occluder selection and hierarchies to naturally support dynamic scenes. Different from prior work, the decoupling of culling and rendering makes our approach practical, and easy to integrate into existing engines. Further, the image-based nature and the possibility of batch-issuing occlusion queries make our solution *scalable*, and lead to *stable* cost, which is fundamental for a fluent user experience.

Our second application is soft-shadow mapping. Typically, a reference soft-shadow mapping renders shadow maps for each light-source sample, leading to significant costs. Instead, we efficiently synthesize these shadow maps. Our strategy significantly improves rendering performance, while maintaining high quality.

Our major contributions, including the two applications, can be summarized as:

- an efficient *backward* warping solution to generate *depth* buffers for novel views;
- a tight search bounding method to efficiently seed the fixed-point search without forward mapping;
- a high-quality depth inpainting strategy suitable for real-time applications;
- an efficient image-based occlusion query technique using our depth warping;

2 RELATED WORK

Image warping has been extensively studied. The methodologies range from efficient to high-quality albeit costly processing. A large body of literature exists in image processing and computer vision, but our focus lies on efficient techniques for rendering, in particular, GPU-based warping. We categorize previous techniques into temporal coherence, multi-source, and single-source warping techniques.

2.1 Temporal Coherence and Reprojection

One fundamental reprojection technique is the reprojection cache. This technique maintains a temporal cache, and reuses previous shading values [Nehab et al. 2007; Sitthi-amorn et al. 2008]. The method exploits temporal coherence between consecutive frames, and can drastically reduce shading costs. Similarly, multi-sample techniques, such as soft shadows and ambient occlusion [Mattausch et al. 2010; Scherzer et al. 2007, 2009], can be accelerated by keeping a history of shadows or occlusion factors. However, these methods require at least one additional render pass with all geometry, which is an important overhead that affects rendering performance. We refer the reader to [Scherzer et al. 2012] for a complete survey.

2.2 Multi-Source Warping

Color-image warping has traditionally been used for morphing, view interpolation, or image-based rendering [Beier and Neely 1992; Chen and Williams 1993; McMillan and Bishop 1995; Seitz and Dyer 1996; Vedula et al. 2002], often relying on correspondences between features of multiple images. The techniques synthesize or interpolate existing images via reprojection, typically involving a small number of views. They have also been applied to sampling plenoptic functions [Gortler et al. 1996; Levoy and Hanrahan 1996].

Recently, warping was used on high-refresh-rate displays [Didyk et al. 2010a]. Using the distinction of I- and B-frames in video compression, they constructed the motion fields from I-frames, and relied on forward warping using a coarse grid for intermediate frames. The bidirectional flow-field definition was later improved for B-frames, yielding higher quality [Yang et al. 2011].

Multi-source depth-buffer warping has hardly been used in real-time rendering, and is more common in disparity-map generation for multiview-stereoscopic videos or TVs [Kang and Ho 2010; Seales et al. 1999; Zinger et al. 2010]. Similar to the multi-source color image warping, the techniques also used feature correspondence across multiple views to recover unknown depths. The solutions are rather slow and require bidirectional reprojection, which inherently delays display by one frame, leading to a non-trivial latency.

2.3 Single-Source Warping

Single-source warping, like ours, is more challenging than multi-source warping, because more disocclusions occur. Regarding forward-mapping approaches, Didyk et al. [2010b] extended temporal up-sampling (with adaptive grid refinement) to stereo-view synthesis. While the warping itself is simple and only one-dimensional, the adaptive refinement often requires high tessellation rates, which reduces performance.

Backward mapping, which is closest to our work, requires a complex warping function and needs to handle many disocclusions. A

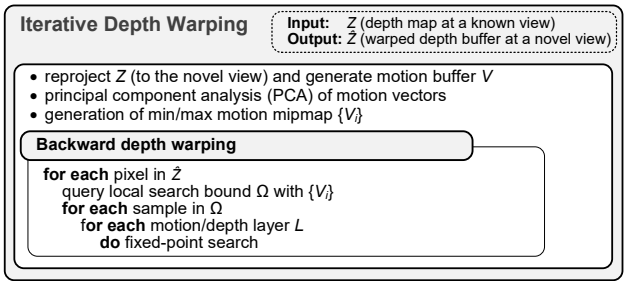

Fig. 2. Overview of our depth-warping framework. For multi-view depth buffers, these steps are repeated as many times as the number of views.

simple yet effective solution is presented by Andreev [2010]. His single-source warping uses motion vectors for the warping, and re-renders foreground objects after replicating neighboring image patches into the background. Recently, Bowles et al. [2012] improved backward mapping via the fixed-point iteration used in [Yang et al. 2011] and a rigorous treatment of convergence conditions. We adapt the fixed-point iteration to support efficient depth warping and investigate a depth-inpainting strategy.

3 ITERATIVE DEPTH WARPING

Backward mapping mostly outperforms forward mapping (e.g., vertex scattering [Lee et al. 2008; Zwicker et al. 2002]). However, if the source point does not lie within a narrow window of its origin, a costly local search is needed. Our work combines a recent fixed-point iteration [Bowles et al. 2012] with a new efficient search-bound tightening. Fig. 2 shows the pipeline of our depth buffer warping.

3.1 Preparation of Depth and Motion Buffers

We use a temporal warping to produce a new depth buffer from the previous frame's depth buffer, and potentially, a spatial warping to obtain several views of this depth buffer in multi-view scenarios. As input, we require the previous depth buffer and the screen-space motion vectors. In the case of dynamic scenes, the item buffer (indices of objects) is also required to apply per-object transformations. After each warping, the produced depth buffer can be used for an early geometry culling, which reduces the rendering cost for the new frame. The details of this culling step will be explained in Section 5. In consequence, after rendering the scene, we obtain a perfect depth buffer (and potentially additional buffers in the context of deferred rendering) as a side product, which is then be used as input to the next depth warp.

Deriving a motion-vector buffer V (combining camera and object motion) has a negligible overhead. It helps us predict the depth buffer of a novel view ($\hat{Z}$) from the source view (Z). 3D Motion vectors are often computed via finite differences [Bowles et al. 2012; Rosado and Studios 2007], which can cause prediction errors with the under- or over-estimation. Instead, we generate a pixel-accurate motion in the image space. Compared to 3D-world displacements, such pixel-unit vectors are more consistent, have better accuracy, and only two components. We compute this 2D motion as follows; given a pixel in the source view $\mathbf{q}$, we compute its new projected position in the novel view $\hat{\mathbf{q}} := C\mathcal{P}^{-1}\mathbf{q}$ by involving the transformation-projection

```
vec3  single_layer_seed( vec2 q ){ return texture(q); } // first layer
float fixed_point_search( vec2 p, vec2 v0 )
{
    vec2 q = p-v0;                  // search seed position
    vec3 s = single_layer_seed(q);  // (2d motion vector, depth)
    for( int k=0; k<MAX_ITER && s.z<1; k++ ){ // s.z<1 means valid
        vec2 w = p-s.xy; if(distance(q,w)<threshold) return s.z;
        s = texture(q=w);
    }
    return 1; // return invalid depth (=1)
}
```

Fig. 3. Pseudocode of the fixed-point search for a single search sample.

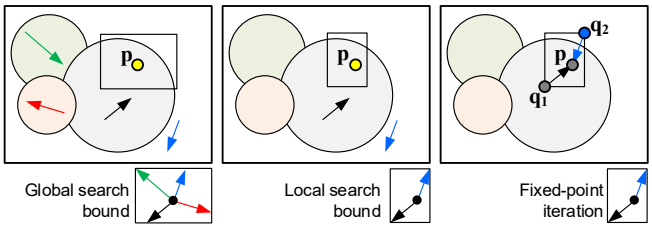

Fig. 4. Warping with fixed-point iteration. We find a conservative search region, tighten it via the motion-vector hierarchy, and search within this region for candidates projecting to the original location. The one with resulting minimal depth is ultimately chosen (here, $\mathbf{q}_1$).

matrices of previous $\mathcal{P}$ and current view C. The 2D motion of $\mathbf{q}$ is then $\mathbf{v}(\mathbf{q}) := \hat{\mathbf{q}} - \mathbf{q}$, which is stored in a render target. In addition to the motion vectors, the render target also stores the new depths, which are assigned during the warping. We note that even camera motions of purely zooming in/out yield 2D motions, due to the perspective projection, except for the center of projection.

3.2 Backward Depth Warping with Fixed-Point Iteration

To efficiently warp a depth buffer, pixel correspondences from the novel view to the (available) source view are needed. Starting with a pixel $\mathbf{p}$ in the novel view, the goal is to find a corresponding pixel $\mathbf{q}$ in the source view, which will move to $\mathbf{p}$. The fixed-point method [Bowles et al. 2012] uses the following iteration: $\mathbf{q}_{i+1} \leftarrow \mathbf{w}(\mathbf{q}_i) := \mathbf{p} - \mathbf{v}(\mathbf{q}_i)$, where $\mathbf{w}$ is the warped position, and $\mathbf{q}_0$ is a starting point (seed) for the iteration. The iteration stops at a fixed point $\tilde{\mathbf{q}}$. As a fixed point, it satisfies $\tilde{\mathbf{q}} = \mathbf{w}(\tilde{\mathbf{q}})$, which implies $\tilde{\mathbf{q}} + \mathbf{v}(\tilde{\mathbf{q}}) = \mathbf{p}$ (i.e., $\tilde{\mathbf{q}}$ moves to $\mathbf{p}$ in the novel view).

If convergent, the method requires only few (e.g., 2–3) iterations, but the seed $\mathbf{q}_0$ is crucial. Straightforward choices, like $\mathbf{q}_0 = \mathbf{p}$, may fail at boundaries between very different motions. Further, at overlaps, hidden points might be chosen instead of visible pixels. Heuristics [Bowles et al. 2012; Yang et al. 2011], such as using a fixed offset, may alleviate symptoms. A better convergence can be guaranteed with the 2×2 Jacobian matrix of $\mathbf{w}$ [Bowles et al. 2012]. Nonetheless, it involves rather costly pre-warping, subdivision steps to split the image along slope discontinuities, and even an additional forward mapping [Bowles et al. 2012].

We propose a simpler yet more efficient solution (Fig. 4); we derive a compact and conservative search region Ω containing the source pixel. Choosing seed points randomly (stratified sampling)

```
vec4 local_motion_bound( vec4 b ) // b=(Left, Bottom, Right, Top)
{
    vec4 m=(1,1,-1,-1)*FLT_MAX; // clear bound for LBRT
    int LOD = ceil(max(0,log2(max(b.z-b.x,b.w-b.y))));
    vec2 p[4] = {b.xy, b.xw, b.zy, b.zw}; // LB, LT, RB, RT corners
    for(int i=0; i<4; i++){
        vec4 s=texture(p[i],LOD); if(!is_valid_motion(s)) continue;
        m=vec4(min(m.xy,s.xy),max(m.zw,s.zw));
    }
    return m;
}
vec4 motion_bound( vec2 p )
{
    vec4 b, m=texture(vec2(0),MAX_LOD); // global motion bound
    for(int k=0; k<MAX_BOUND_ITER; k++){ // typically, 2--4
        b=m; vec4 m=local_motion_bound(vec4(p-b.zw,p-b.xy)); // LBRT
        if(!is_valid_motion(m)) return b; // usually background
    }
    return m;
}
```

Fig. 5. Pseudocode of the iterative bound tightening process.

within Ω is usually enough to ensure convergence. Our solution might even detect multiple candidates and we keep the one with resulting minimal depth. Hereby, we can handle many ambiguous cases. If convergence still fails, we propose a multi-layer inpainting solution (Sec. 3.4). Fig. 3 shows the pseudocode for our approach.

3.3 Search Region Bound for Motion Vectors

To find the compact search region Ω to initialize the seeds, we build a motion-vector hierarchy $\{V_i\}$ using *min-max mipmaps* [Williams 1983]. Let V_i represent level i, and V_0 the original motion buffer. A texel $\mathbf{q}$ of V_i holds the minimum and maximum xy components of the 2D motion vectors within a square area of $2^i \times 2^i$ pixels. The four coordinates are encoded in a single texture. Consequently, the texel on the highest mipmap level holds a global bound for all motion vectors, which can be used as a first conservative estimate for Ω. To narrow down Ω, we perform a series of mipmap lookups, each reducing the search area (see Fig. 5 for the pseudocode). For a region of size $a \times b$, we first choose the mipmap level $\lceil \log_2 \max(a, b) \rceil$, and perform four lookups, one for each corner of the region. These lookups always conservatively cover the original region. Taking the extrema of the four texel values from V_i delivers the new bounds on the motion and defines a new Ω. In practice, we found that two to three such narrowing steps suffice to reduce Ω significantly.

Relying on mipmaps and four lookups to cover Ω might seem coarse, but provides a good tradeoff; better bounds for motion vectors did not result in higher quality. Instead, an improvement can be reached by using a tighter bound Ω to choose seeds from. To this extent, we align the axes of the view to obtain a better fit for an axis-aligned bounding box (AABB) [Gottschalk et al. 1996]. The gain stems from high correlation of motions; e.g., when moving the camera, many static objects will reflect the same motion direction. To decide on the axes, we perform a principal component analysis (PCA) of the 2D motion vectors. The PCA does not require involving all the vectors, and a small number (e.g., 64) usually suffices for this purpose. We then apply the change of the coordinate system, before constructing the aforementioned min-max mipmap representation.

```
vec3 multi_layer_seed( vec2 q, int layer )
{
    for( int j=layer; j>=0; j-- ){
        vec3 s=texture(q,j); if(s.z<1) return s; // a valid seed
    }
    return vec3(0,0,1); // the invalid seed
}
```

Fig. 6. Pseudocode of the cross-layer seeding for the multi-layer search.

When reading the bound, reversing the motion coordinates (i.e., a rotation by 2×2 matrix) recovers the motion bound.

The retrieval of the tight motion bound improves quality, given the same number of search seeds, and is also useful to adaptively sample the search seeds. The local motion bound lying in a narrow area implies that the search region is homogeneous. This case does not require many samples, and an early exit can be employed. This simple strategy leads to a significant speedup by adding more seed samples only, where we encounter complex configurations.

3.4 Multi-Layer Warping for Depth Hole Filling

While a successful iteration process results in a good depth buffer value in a novel view, if the necessary seed for a region was not found, there might still be holes. In particular, hidden surfaces, which novel views may reveal, cannot be warped. In this section, we present a high-quality depth inpainting strategy relying on *deep depth buffers*, which distinguishes ours from common approximations (e.g., hallucinations).

To generate a deep depth buffer, we employ depth peeling [Everitt 2001], which successively extracts depth layers. In each iteration, only the geometry behind the previously extracted depth buffer is rendered. Hereby, layers are successively peeled off the scene. By warping all layers into the new views, holes can be avoided as occluded geometries are involved, maintaining high quality. Typically whenever the number of novel views exceeds the number of layers (e.g., soft-shadow mapping), the approach becomes efficient.

For multi-layer depth warping, we additionally repeat the depth warping for each hidden layer to handle disocclusions. This is straightforward, but does not necessarily lead to high benefit. The iterative search may easily end up in an empty region, where the iteration cannot proceed further; hidden depth layers are typically sparse and exhibit wide empty regions.

To improve the process, the search should continue across multiple layers. However, considering all layer combinations would be too costly. Instead, we launch one search per layer, and generally do not move across layers. Only when reaching an empty region of a layer, we move the point upwards in the layers towards the camera until the location is non-empty; Fig. 6 shows the pseudocode of this cross-layer seeding process. If our search during the iterations ends twice in an empty area, we consider the sample as non-convergent.

This strategy effectively improves the quality of the multi-layer depth warping. Moreover, when there are large background objects (e.g., ground/floor), adding them as a separate (the last) depth layer is helpful in widening their connected search areas; this additional layer can be rendered at negligible cost.

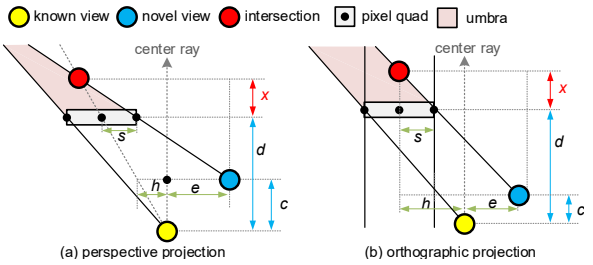

Fig. 7. Generalized umbra culling for non-planar camera displacements. Objects in the umbra are not visible from the novel view.

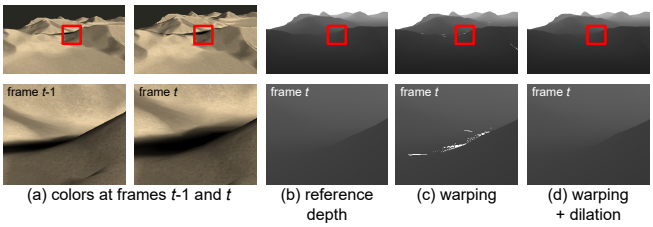

Fig. 8. Geometric edges might invert their screen-space topology, which might not be captured with depth peeling (a). This may lead to depth holes in warping (c), which can be approximately filled by dilating neighbors (d).

Generalized Umbra Culling. A complex scene may require many depth layers to completely fill all holes due to the warping. In this case, depth peeling becomes too costly. To reduce the number of layers, we revisit and improve *umbra culling*, which was previously used for lens-blur rendering [Lee et al. 2010].

Umbra culling back-projects a pixel to 3D, and interprets it as an opaque quad geometry. The volume behind this quad, which is not reached by any rays from the novel views, is referred to as the *umbra*. Fragments in the umbra are not visible from the novel views, and therefore, can be safely ignored when warping the depth buffer to those views. In our context of depth peeling, fragments whose sampling points are within the umbra can be ignored. While the original idea was applied only for planar displacements on a camera aperture, we here generalize the idea towards arbitrary camera motions, including orthographic projections (used for shadows of directional lights). Specifically, the threshold x, which describes the extent of the umbra is given via:

$$x := \frac{(d-c)s}{(e+h)-s}, \tag{1}$$

where d and s indicate the depth and size of the pixel quad; c and e are the depth of the novel camera and its distance to the center ray; and h is the offset from the center ray to the projector ray at depth c. Standard depth peeling uses $x = 0$, whereas the original umbra culling is obtained by setting $c = 0$, $h = 0$ (i.e., $x := ds/(e-s)$).

Fig. 7 illustrates all the parameters. The umbra culling ensures a faster progress, due to the positive depth offset x that is added to each layer. Moreover, when $(e+h) < s$, the number of required layers is *bounded*, enabling to predict a maximum number of depth layers. The number of layers for standard depth peeling is only bounded by the scene geometry, which can be arbitrarily complex.

Fig. 9. The four scenes used for experiments: Yeahrights (YR), Monsters (MO), Balloons (BL), and Cityblock (CB). YR model is provided courtesy of Keenan Crane, MO models {shiva3d|GalaxyArt|Pedro Barbaro} at turbosquid.com, and CB models {fingerz|digitalstonemason} at 3dwarehouse.sketchup.com and Reallusion at reallusion.com, respectively.

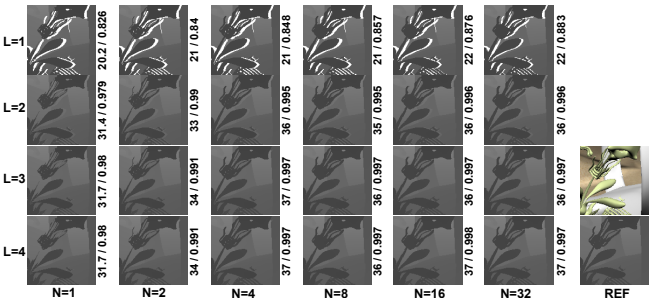

Fig. 10. Effects of the search seed samples (N) and the number of layers (L) on quality. The numbers on the right of each image indicate PSNR (dB) and SSIM values, measured with the reference depth-buffer rendering (REF).

Handling Degenerate Cases. Depth peeling based on a single view, fails to capture surfaces in certain cases, such as a geometry whose normal is orthogonal to the ray from the view point. A worst case is when back-facing triangles at the known view (not rendered with back-face culling) are inverted to front faces in the novel view. Such cases occur on rare occasions at the silhouettes/edges of large objects (e.g., ground/terrain), and may lead to depth holes (Fig. 8).

While it would be possible to detect triangle inversions and treat these triangles separately, the rarity of these cases does not warrant higher computational costs. We propose a practical solution based on the observation that the problem mostly occurs with rapid motion (e.g., hundreds of pixels per frame). When a hole in the depth map is surrounded by successful searches, we close the hole by propagating the nearest neighbor's depth via a dilation (Fig. 8). In practice, this strategy performs well, and leads to a good approximation.

4 EXPERIMENTAL ANALYSIS

This section reports our experiments to assess our depth warping solution in terms of accuracy and performance.

4.1 Method

We implemented our depth warping in OpenGL 4.5 on an Intel Xeon E5-1620 v2 with 3.7 GHz and NVIDIA GeForce GTX 1080. All tests were performed at full-HD resolution (1,920×1,080).

Four scenes (Fig. 9) are tested with camera movements. The Yeahrights scene (YR) is a simple scene with a small number of

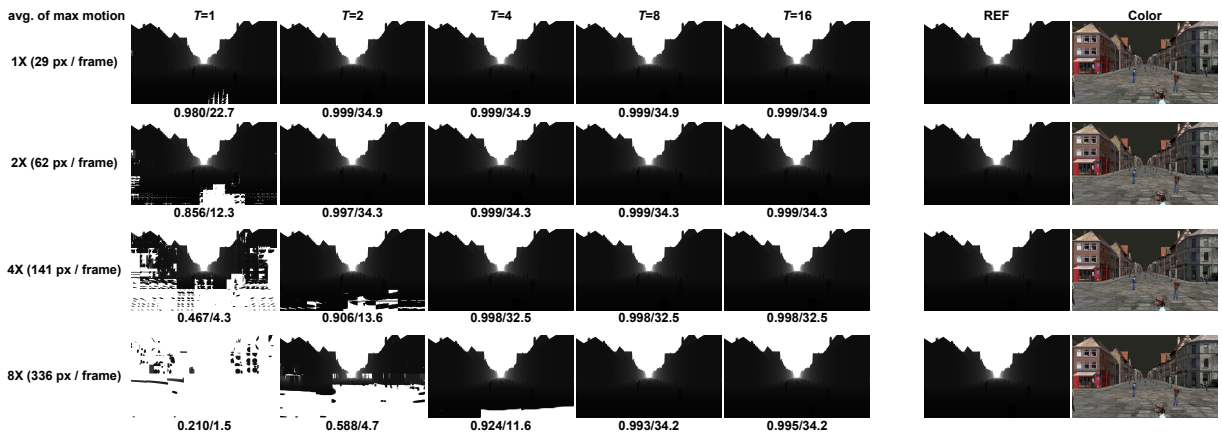

Fig. 11. Effects of the number of iterations (T) against motion speed. The motion speed indicates the averages of the maximum motion per each frame for animated sequences (8, 4, 2, and 1s, respectively). SSIM and PSNR are shown for the particular frame here, and $L = 4$ and $N = 8$ were used.

objects, but with a high polygon count and high depth complexity. The Monsters scene (MO) is a game-like scene of medium complexity, with around 6.3K objects. The Balloon scene (BL) contains around 35K objects (balloons with tiny baskets), resulting in high complexity in terms of depth. The Cityblock scene (CB) is an urban-like example with a large number (around 94K) of objects. All scenes used a pre-defined camera movement of 7–13 seconds; see the accompanying video clip for the sequences. The averages of maximal per-frame motion in the scenes are 16, 35, 26, and 29 pixels/frame for YR, MO, BL, and CB, respectively; we also report the effects of stronger motion.

We compare depth buffers produced with our warping solutions against the depth-only model rendering as a reference depth buffer (REF). Our experiments vary the number of search seeds (N), the number of input depth layers (L), the number of iterations for each search seed (T), and the magnitude of per-frame motion vectors. While the single-layer solution ($L = 1$) keeps holes (filled with the background depth), the multi-layer solutions ($L > 1$) use deep depth buffers, and employ our inpainting. When a terrain or ground is available (YR, MO, and CB), the last layer was peeled without foreground objects. For the quality/performance comparison, we used the oriented tight motion bounds unless explicitly noted, and disabled our dilation solution to solely illustrate the depth warping effects.

For additional comparison, we implemented two GPU-based per-pixel forward-warping techniques. The first is a vertex-driven forward warping (VFW) [Lee et al. 2008; Zwicker et al. 2002]. We define pixel-size quads as many as the image resolution, and splat the four corners of each quad by their motion vectors with depth buffering. The second is based on the recent atomics-driven forward warping [Cichocki 2017; Doghramachi and Bucci 2017; Yu et al. 2010] (AFW), which directly writes source vertices to target positions with *atomic min* operation to ensure depth test. For multi-layer warping, we adapt VFW and AFW to repeat them for each layer. However, as indicated by Yu et al. [2010], both techniques suffer from seams/-holes even in smooth surfaces, when a single quad/pixel maps to a larger area. To alleviate this problem, we extend the support size of quad/vertex [Yu et al. 2010] (in our case, by a half-pixel length to each side); though for large areas, AFW still exhibits holes.

4.2 Depth Warping Accuracy

We first report the effects of N and L together (Fig. 10), which are the major factors affecting the quality of depth warping. Peak signal-to-noise ratio (PSNR) and structural similarity [Wang et al. 2004] (SSIM) are also reported as quantitative quality metrics, measured against REF. For all Ls, most of the areas quickly converge with a small number of samples (3–4), but at complex junctions, more samples are needed. The single-layer warping ($L = 1$) improves with more samples, and is saturated around $N = 32$. The samples that converge in this case lead to good local quality, but overall, too many pixels fail to converge, and large holes occur. In contrast, the multi-layer warping ($L > 1$) saturates much faster and only a small number of pixels fail to converge. Fig. 10 shows that most pixels converge, and a high quality (> 35 dB and 0.995 in terms of PSNR and SSIM) is reached, making the difference almost imperceptible. The SSIM is particularly high, as failures mostly occur on sub-pixel level. When the number of samples exceeds 8 and more than two layers are used, we obtain a near-perfect depth buffer even for rapid motion (e.g., up to 100 pixels/frame of motion). Four layers are sufficient even for very complex scenes, but it entails more computational costs. In practice, 2–3 layers are a good tradeoff.

Table 1 shows the quality comparison of VFW and AFW with ours ($N = 16$). For both layer configurations ($L = 1$ and $L = 4$) and all the scenes, our solution always has higher accuracy than the other forward warping solutions. This accuracy results from the support-size expansion for both VFW and AFW, slightly dilating the boundaries; without the expansion, the qualities are much worse. Thereby, their PSNR differences are higher than their SSIM differences. In particular, AFW has a much lower accuracy than the others (down to 8.8 dB of PSNR against ours), because AFW still exhibits erroneous seams/holes, despite support-size expansion.

We also examined the effect of the number of iterations (T) versus motion speed. In practice, T is less important (2–3 iterations suffice) than N and L, but the quality is often affected for rapid motions.

Table 1. Quality/performance comparison of the vertex-driven forward warping (VFW) and atomics-based forward warping (AFW) with ours ($N = 16$). The quality is measured against REF in terms of SSIM/PSNR (no unit/dB), and performance in terms of the frame time (measured in ms).

	Ours ($L = 1$)		VFW ($L = 1$)		AFW ($L = 1$)		Ours ($L = 4$)		VFW ($L = 4$)		AFW ($L = 4$)	
	quality	time	quality	time	quality	time	quality	time	quality	time	quality	time
YR	0.968/27.5	0.7	0.934/23.3	2.2	0.950/24.3	0.3	0.996/36.1	0.8	0.997/28.2	6.3	0.993/27.9	0.4
MO	0.992/30.3	0.6	0.982/26.0	2.1	0.983/25.8	0.3	0.998/34.8	0.8	0.998/30.0	6.2	0.995/28.8	0.4
BL	0.954/23.3	0.8	0.935/21.1	2.2	0.945/21.2	0.3	0.994/29.4	1.3	0.993/26.7	6.1	0.985/25.1	0.6
CB	0.994/31.3	0.6	0.979/25.5	2.1	0.985/25.7	0.3	0.999/36.7	0.8	0.998/27.6	6.1	0.998/27.9	0.5

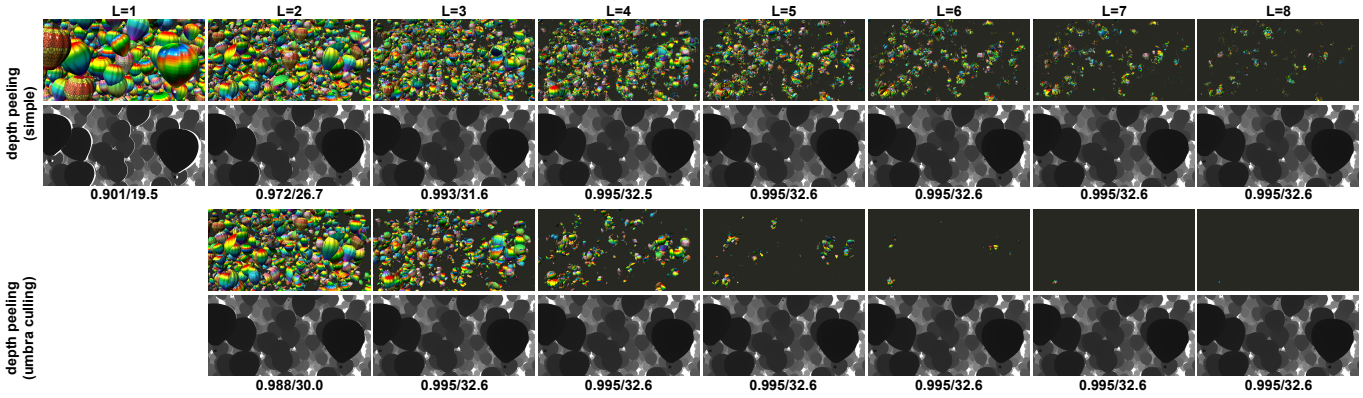

Fig. 12. The effect of umbra culling for generating deep depth buffers, assessed in terms of SSIM/PSNR differences (no unit/dB). The umbra culling peels effective layers more quickly, and thereby, the quality of the warping (here, $N = 16$) becomes similar with a reduced number of layers.

Table 2. Quantitative quality improvements by the tight motion bound from the global motion bound, assessed in terms of SSIM/PSNR differences (no unit/dB) for the MO scene.

L	$N = 1$	$N = 2$	$N = 4$	$N = 8$	$N = 16$	$N = 32$
1	0.04/4.66	0.02/2.63	0.01/1.24	0.01/0.91	0.01/1.26	0.01/1.46
2	0.04/6.03	0.02/3.84	0.01/1.95	0.00/1.00	0.01/1.18	0.01/1.46
3	0.04/9.48	0.02/6.17	0.00/2.56	0.00/0.54	0.00/0.37	0.00/0.28
4	0.03/9.68	0.02/6.54	0.00/2.64	0.00/0.58	0.00/0.12	0.00/0.11

Fig. 11 compares different values of T against various magnitudes of motion. The pre-recorded movements of the CB scene were scaled up to 8×. As illustrated, $T = 2$ is already sufficient for a scaled motion of 1–2×. As for the 4× motion, $T = 4$ is required to reach saturation. Here, the quality is marginally lower than 1–2×. The 8× motion requires more than $T = 8$ to converge without depth holes in flat regions. Note that these exaggerated motions were only used for demonstration purposes. For practical use, 2–3 iterations suffice.

We also report on the effects of the tight search bound, generated with our mipmap-based tightening process (Sec. 3.3). Table 2 shows quantitative quality improvements of the tight motion bound against the global bound. The tight search bound results in significantly higher accuracy than the global motion bound for all the test scenes. The improvement is pronounced, when N is small (up to 9.68 dB).

The oriented bounds (OBs) of the motion vectors can improve the quality when using the same search seed samples, while requiring negligible constant cost (e.g., 0.2 ms for 64 objects) for the PCA analysis. However, the OBs do not necessarily end up with smaller areas (i.e., denser sampling) than the axis-aligned bounds (AABs). We experimentally assessed the accuracy of OB against AAB for the animated sequences, and found that the OB results in higher accuracies for 66.7, 50, 41.7, and 50 % of the combinations of $L \in \{1, 2, 3, 4\}$ and $N \in \{1, 2, 4, 8, 16, 32\}$ for YR, MO, BL, and CB scenes, respectively. When the area of OB is reduced, the qualities are slightly enhanced by up to 0.007 and 1.2 dB in terms of SSIM and PSNR difference. In practice, we can always use a tighter bound by selecting the one that has smaller area for every frame.

Fig. 12 shows our umbra culling for depth peeling, and compares simple depth peeling (constant thresholding) and our solution. The color images in the upper rows visualize peeled layers, while the lower row shows the warped depth buffers using the corresponding layers. Umbra culling peels layers significantly faster than standard depth peeling; when $L > 8$, the layers are almost empty, reaching the upper bound for this scene. When $L > 4$, the quality of both peeling techniques are similar; but for less layers ($L \leq 4$), umbra culling improves the quality. In this example, standard depth peeling with $L = 5$ is equivalent to the quality of umbra culling with $L = 3$. In practice, 2–3 layers are sufficient with umbra culling to achieve high quality at low cost.

4.3 Depth Warping Performance

To assess performance, we assume that depth and motion buffers are readily available (common in many deferred rendering pipelines).

Table 3. Performance comparison (measured in ms) of our depth warping with tight local and global bounding and motion mipmap construction (MMC), assessed in terms of the number of search seed samples (N) and the number of layers (L). Each search seed was iterated 3 times ($T = 3$). The reference (REF) indicates a depth-only geometry pass for comparison.

Scene	L	Ours (ms)													REF (ms)
		Depth warping (tight bound)						Depth warping (global bound)						MMC	
		$N=1$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=1$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$		
YR	1	0.26	0.28	0.31	0.38	0.53	0.81	0.12	0.17	0.27	0.48	0.88	1.66	0.12	1.44
	2	0.27	0.29	0.33	0.43	0.61	0.95	0.13	0.19	0.30	0.52	0.94	1.79	0.14	
	3	0.27	0.29	0.35	0.46	0.65	1.05	0.13	0.19	0.31	0.54	0.99	1.88	0.14	
	4	0.27	0.30	0.35	0.47	0.70	1.12	0.14	0.20	0.33	0.57	1.04	1.98	0.14	
MO	1	0.27	0.28	0.31	0.36	0.48	0.73	0.12	0.17	0.28	0.49	0.90	1.72	0.12	3.21
	2	0.27	0.29	0.32	0.40	0.55	0.87	0.13	0.18	0.29	0.51	0.95	1.81	0.13	
	3	0.28	0.30	0.34	0.44	0.63	1.02	0.13	0.19	0.31	0.54	1.00	1.92	0.14	
	4	0.28	0.30	0.35	0.45	0.66	1.08	0.14	0.20	0.39	0.91	1.84	2.04	0.14	
BL	1	0.30	0.31	0.35	0.45	0.65	1.09	0.14	0.23	0.32	0.60	1.01	1.95	0.13	15.78
	2	0.31	0.36	0.46	0.64	0.99	1.66	0.14	0.21	0.35	0.61	1.13	2.17	0.15	
	3	0.32	0.37	0.47	0.67	1.05	1.79	0.15	0.23	0.39	0.68	1.25	2.38	0.17	
	4	0.32	0.38	0.49	0.71	1.12	1.87	0.16	0.24	0.41	0.75	1.33	2.54	0.18	
CB	1	0.27	0.28	0.30	0.34	0.43	0.62	0.12	0.18	0.30	0.53	0.97	1.82	0.12	37.52
	2	0.27	0.29	0.32	0.40	0.55	0.87	0.13	0.19	0.31	0.55	1.01	1.93	0.14	
	3	0.27	0.29	0.34	0.42	0.59	0.96	0.13	0.19	0.32	0.56	1.03	1.97	0.15	
	4	0.28	0.30	0.34	0.43	0.64	1.05	0.13	0.19	0.33	0.56	1.05	2.03	0.16	

Even in the case that the motion buffers are not available, it is marginal to generate them (in our implementation, it took 0.1 ms for each layer in the full-HD resolution). When particular objects are dynamic, an item buffer (having their transformation IDs) is also used to integrate object and camera motions; the item buffer can also be produced during deferred rendering. We evaluate the performance of depth warping and motion mipmap construction (for the motion bound query), excluding setup costs and cost for the basic rendering pipeline. To illustrate the difference, the depth warping performance is reported for both the tight local bound and global bound. As umbra culling is used during depth peeling, it has no impact on the performance of the warping. For complete timings, please refer to the applications (Secs. 5 and 6).

Table 3 summarizes the performance benchmark. The performance of our solution scales with the number of search seed samples (N) and the number of layers (L). All tests used 3 iterations ($T = 3$) for the search. The table shows that the performance of our solution is stable, and less dependent on the geometric complexity of the scene. A dependency on depth complexity still exists (a simple scene converges faster), but to a limited amount. The time complexity of N is sublinear, as flat regions are skipped after a small number of samples (1–2). The dependence on L is also sublinear, due to early skipping of empty areas on the hidden layers. The tight bound is more costly, when $N \leq 4$ due to the additional lookup to the motion mipmap (here, roughly 0.14 ms); but, as N grows ($N > 4$), it starts to greatly reduce the search cost with faster convergence; the speedup factors for $N = 32$ are 1.3–2.9×. The motion mipmap construction, required to query the local motion bounds during the warping for each pixel, is negligible (typically less than 0.15 ms in our experiments) for a resolution of 1,024×1,024, which is the nearest power-of-two-size below full HD.

Table 1 shows the performance comparison of VFW and AFW with ours ($N = 16$); ours include the timings for motion-mipmap generation. On average, our solution is 3.2× and 7.0× faster than VFW for $L = 1$ and $L = 4$, respectively, and around 2× slower than AFW for $L = 1$ and $L = 4$. VFW requires drawing many points/triangles; for example, VFW with $L = 4$ splats 16M triangles, which is very costly. AFW is cheaper, due to the efficient atomics-based writing, avoiding the quad rasterization for splatting. However, AFW significantly trades quality for performance, because it cannot handle large holes in smooth surfaces and discrete boundaries well. This is why our backward warping results in higher quality, and the additional cost of ours to AFW is small (here, 0.4 ms).

Light-weight scenes are cheap to render, which leads to little gain when using warping. However, with growing geometric complexity, warping becomes increasingly effective. Using our recommended setting of $N = 4$ and $L = 3$, the speed-up of depth warping against depth-only rendering is 2.9, 6.7, 24.7, and 76.6 for YR, MO, BL, and CB, respectively. As an additional improvement, the depth-map resolution can be reduced for some applications, where an additional significant speedup (e.g., up to 2.5× for a quarter-size resolution) is achieved, since most steps of our solution are image-based.

Regarding the depth-peeling process, while the foreground depth layer is usually available, additional layers need to be extracted. Fortunately, as the sparsity of the output increases, the cost per layer shrinks, and falls slightly below direct rendering. Table 4 shows the cost for the depth peeling.

Table 4. Rendering cost (measured in ms) of the depth peeling, assessed in terms of the number of layers (L). For multiple layers, the last layer peels only the background, except for the BL scene (having no ground).

Scene	L =1	L =2	L =3	L =4
YR	1.51	1.64	3.14	4.54
MO	3.46	3.58	6.94	10.28
BL	16.46	32.10	48.27	63.89
CB	37.74	37.97	74.50	110.94

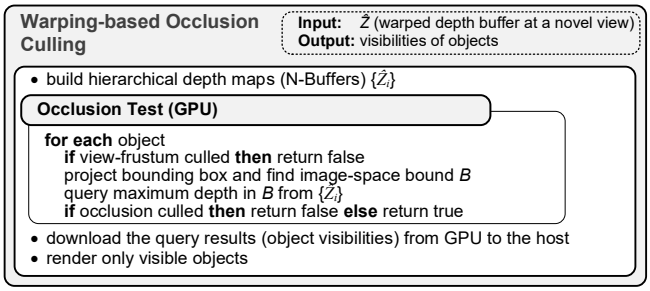

Fig. 13. Overview of our occlusion culling framework.

5 APPLICATION: OCCLUSION CULLING

Visibility culling is a fundamental tool for real-time rendering. It does not degrade image quality and removes invisible content. View frustum culling (VFC) is an efficient example, where objects are tested against the frustum, and ignored if they fall outside. Occlusion culling additionally considers objects hiding each other.

This section introduces the first application of our depth warping, *warping-based occlusion culling* (WOC). Fig. 13 gives an overview of our solution. Our *single-layer* depth warping (with depth holes) can be seamlessly integrated for high-performance occlusion culling.

We revisit the hierarchical occlusion mapping (HOM), which is one of the specialized algorithms to efficiently query occlusion maps. Usually, potential occluders are selected and rendered. The resulting depth buffer (*occlusion map*) is used to verify the visibility of the remaining *occludees* (or their bounding proxies). Efficiency highly depends on the occluder choice, but a precise selection is costly, and heuristics (e.g., the projected area) are usually preferred. Our method does not require occluder/occludee distinctions, and uses all visible elements of the previous frame; this is a near-optimal choice with (almost) perfect occluder fusion [Wonka et al. 2000]. However, instead of rendering actual objects, we use them *implicitly* by *warping the previous depth buffer* to the new frame.

5.1 Previous Work and Background

Typically, the occlusion test uses hardware occlusion queries (HOQs) executed on GPUs [Cunniff et al. 2001], which deliver the amount of potentially drawn pixels. The rendering process can then be steered by reading back the result to the host application.

Such occlusion culling leads to several problems, reducing its utility in practice. First, choosing potential occluders prior to the actual rendering can require metadata, or occluder fusion operations, as well as non-trivial and costly computations for high depth/triangular complexity [Luebke and Georges 1995; Wonka et al. 2000]. Coarse heuristics [Klosowski and Silva 2000; Kubisch and Tavenrath 2014] reduce the overhead, but, more objects are falsely considered visible. Second, reading back query results leads to latency, due to the asynchronous GPU threading model. Even for recent HOQs, which accelerate the test itself, the latency problem persists. So, reducing the number of read-backs is crucial for culling efficiency.

Spatiotemporal coherence combined with hierarchical structures can reduce the amount of queries. Coherent hierarchical culling (CHC) [Bittner et al. 2004] maintains a scene-hierarchy cut (e.g., bounding volume hierarchy), to avoid re-testing many objects. CHC has been improved to integrate hardware-dependent statistics [Guthe et al. 2006], better scheduling and tighter bounds (CHC++) [Mattausch et al. 2008], as well as ray tracing [Mattausch et al. 2015]. Nevertheless, per-object (or small-batch) queries remain necessary, and cause redundancy, irregular stalls, as well as state changes in the rendering pipeline, which reduce performance, and make them hard to optimize or integrate into existing engines. Also, maintaining hierarchical structures for dynamic scenes remains challenging.

More recent GPU-based approaches repeat the pair of culling and geometry passes twice. The first culling tests against the previous depth buffer [Haar and Aaltonen 2015], or the rendering of the previously visible objects [Nießner and Loop 2012]. The second-stage culling is done against the result of the first geometry pass, followed by the rendering of false negatives. Another approach is a hybrid (together with the CPU) pipeline, which reprojects the previous depth buffer, and dilates holes [Kasyan et al. 2011]. For efficiency, a coarse resolution is used for the reprojection and occlusion culling.

Our WOC, implemented entirely on the GPU, improves the existing problems to a great extent. It does not need the selection of occluders and considers all the previously-visible pixels as occluders. Its image-based nature efficiently facilitates batch tests/queries in parallel. The amount of occlusion queries reaches up to hundreds of thousands and we demonstrate its high scalability. Furthermore, the batch queries do not rely on a hierarchy and run in a single GPU pass, which makes its integration into the existing engines less intrusive (i.e., a single geometry pass) and the handling of dynamic objects becomes easier.

5.2 Algorithms

Our approach first employs the standard VFC. Only the remaining objects are tested using a customized query against the predicted depth buffer $\hat{Z}$. Given a screen-space bound computed from the proxy of an occludee, we recover the maximum depth within this bound. When the minimum depth of the proxy corners exceeds the maximum of the screen-space depth, we can safely cull the object.

To efficiently find the maximum depth for a screen-space bound, we could build a mipmap from the warped depth buffer $\hat{Z}$, similar to the motion buffers. Unfortunately, this choice is too approximate. Instead, we rely on *N-buffers* [Décoret 2005]. An N-buffer is a set of textures $\{\hat{Z}_i\}$ of identical resolution, where a texel $\mathbf{p}$ of $\hat{Z}_i$ holds the maximum depth within a square area of $2^i \times 2^i$ pixels, whose lower left corner is located at $\mathbf{p}$. To test a region of size $a \times b$, we use the N-buffer texture $\hat{Z}$ at a level $m = \lfloor \log_2 \min(a, b) \rfloor$, and choose lookup

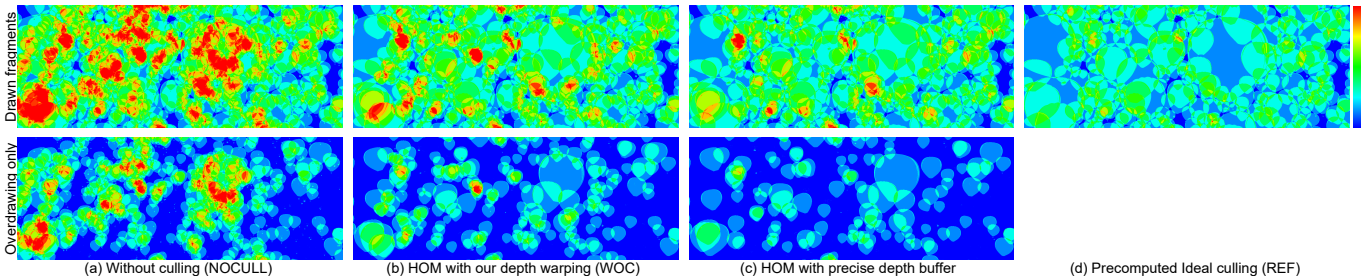

Fig. 14. Comparison of our warping-based culling (WOC) with the precomputed ideal ground-truth culling (REF), hierarchical occlusion mapping (HOM) with the precise depth buffer, and no culling (NOCULL). The upper and lower rows indicate the drawn fragments and overdrawn fragments (with respect to REF), respectively. Blue-to-red in the color mapping indicates low-to-high fragment overdrawing.

Table 5. Performance (frame time measured in ms) comparison of our culling (WOC) against no culling (NOCULL), view frustum culling (VFC), state-of-the-art CHC++, and ideal culling (REF).

Scene	NOCULL	VFC	CHC++	WOC	REF
YR	2.0	1.9	1.8	2.9	1.8
MO	3.8	2.6	4.3	2.5	1.2
BL	14.6	15.4	27.1	4.5	2.2
CB	38.4	43.8	18.3	6.2	4.2

locations for the tile of $2^m \times 2^m$. Overlapping $\lceil a/2^m \rceil \times \lceil b/2^m \rceil$ tiles ensures to accurately cover the entire test region of arbitrary rectangular shapes.

Even using the efficient N-buffer construction, the costs are clearly higher than for mipmaps. A good tradeoff is to use Y-Maps (a combination of both, where the first i levels rely on a mipmap, and from level i on, use N-Buffers) [Schwarz and Stamminger 2008]. Y-Maps with two mipmap levels lead to a lower construction time, without sacrificing much precision regarding the depth queries.

Once the occlusion test is done, the test results are stored in a GPU buffer or texture. To steer the further rendering, the result requires to be downloaded to the host application. This stalls the pipeline, but the cost is relatively low (see results). Further, when the recent extension of indirect batch rendering (reducing the overhead of the GPU driver) is available, we can directly issue a single draw call for all objects, entirely eliminating the stalls.

5.3 Evaluation

We implemented WOC on the same hardware and graphics API. We used the same resolution and scenes from our experiments in Sec. 4. We compare our solution with several methods: ideal culling (REF), simple rendering without any culling (NOCULL), standard view frustum culling (VFC), and state-of-the-art CHC++ [Mattausch et al. 2008]. For REF, we precomputed an ideal set of visible objects. CHC++ used a bounding volume hierarchy of per-object bounding boxes built with the surface-area heuristic. CHC++ included batch, randomization, tighter bounds of inner nodes, and multiqueries, with the following parameters: $n_{av} = 10$, $b = 1000$, $d_{max} = 4$, and $s_{max} = 1.4$ [Mattausch et al. 2008]; we note that b and d_{max} were tuned from the original suggested parameters ($b = 50$ and $d_{max} = 3$) for higher performance in our experimental setup. Our warping solution used a single layer (depth holes are conservatively kept at the far-clipping depth). The number of used search seeds (N) was 8. The geometry rendering used a deferred rendering pipeline, except for CHC++ (alternating between culling and rendering).

Table 6. Performance (measured in ms) breakdown of our WOC.

Scene	Motion generation	Depth warping	Occlusion culling	Total
YR	0.12	0.49	0.30	0.91
MO	0.12	0.47	0.34	0.93
BL	0.14	0.57	0.58	1.29
CB	0.13	0.47	1.43	2.03

Table 7. Comparison of WOC and approximate WOC (AWOC) on culling performance (frame time measured in ms) and quality error. The original animated sequences were sped up from 1× to 16× to scale up the camera motions. The errors are measured as the quality difference (PSNR) between the color renderings of WOC and AWOC.

Scene	WOC (ms)					AWOC (ms)					PSNR (dB)				
	1×	2×	4×	8×	16×	1×	2×	4×	8×	16×	1×	2×	4×	8×	16×
MO	2.5	2.6	2.8	2.9	3.1	2.5	2.5	2.7	2.8	2.9	71.1	60.6	54	49.1	48.2
BL	4.5	4.7	5.1	5.6	6.4	4.4	4.4	4.5	4.7	5.1	48.8	42.8	35.4	31.3	28.9
CB	6.2	6.4	6.8	7.2	8.7	6.1	6.2	6.5	6.6	6.8	∞	∞	61.1	61.5	65.4

Our culling produces hardly perceptible differences compared to REF (60 dB of PSNRs and > 0.999 of SSIMs). A failure of the iterative search during depth warping does not necessarily lead to culling errors, since we keep the depth of such pixels conservatively at the far clipping depth. Only tiny objects (of 1–2 pixels) might be falsely culled, due to rasterization precision.

Table 5 summarizes the measurements. For (relatively simple) YR and MO scenes, VFC suffices, and no sophisticated culling is needed. For high-complexity scenes (BL and CB), our solution significantly outperforms all other techniques. Our speedup over VFC/CHC++ reaches up to 3.4/7.4× (BL scene) and 7.1/4.1× (CB scene). Note that, for the MO and BL scenes, VFC outperforms CHC++, due to the costly switches between rendering and querying.

Our solution outperforms the others mainly due to the absence of hierarchical structures, enabling an efficient GPU parallelization.

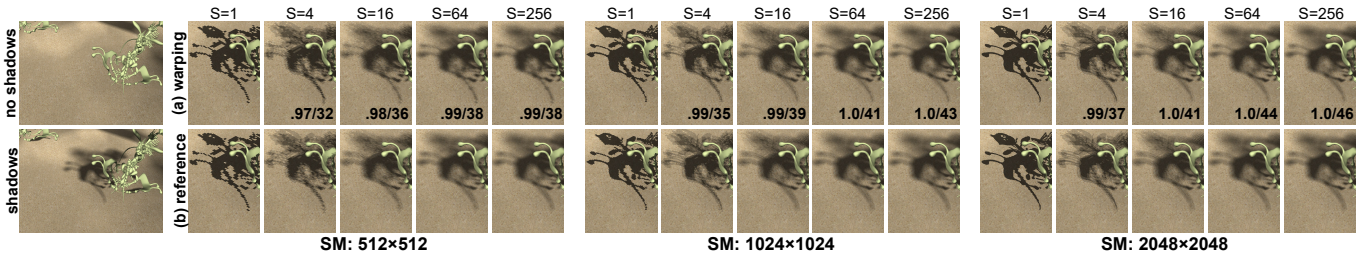

Fig. 15. Comparison of soft-shadows rendering using (a) our depth warping solutions, and (b) reference with respect to the shadow map resolution (SM) and the number of area light samples (S). The quality comparison used SSIM (no unit) and PSNR (dB), and is noted at the bottom of each image of our solution.

Also, the image-based nature leads to more stable culling at constant cost, which is important in time-critical applications. The performance of CHC++ may degrade with an excessive amount of queries and frequent state changes, which can introduce latency and irregular costs. The average number of per-frame queries and state changes of CHC++ are 2.4/3.0, 317.2/9.0, 2847.1/14.8, and 1988.7/16.1 for YR, MO, BL, and CB scenes, respectively. Our culling uses screen-space bounding boxes with potentially lower culling ratios (Fig. 14 compares this to ideal culling (REF) and a precise depth buffer) but the cost for the occlusion tests is significantly lower.

Table 6 shows the timing breakdown of our solution. The generation of the motion buffer and depth warping is nearly constant (< 0.71 ms). Culling costs are 0.30, 0.34, 0.58, and 1.43 ms for the YR, MO, BL, and CB scenes, respectively. Occlusion tests have nearly constant cost but the download time (to the host CPU application) depends on scene complexity and number of objects.

In contrast to many previous hierarchy-based methods, our approach supports dynamic scenes. The overhead to update transformation matrices for dynamic objects is small (e.g., < 0.5 ms for the fully dynamic BL scene). By simulating animation on the GPU, this overhead would shrink even further.

Our WOC works best with moderately coherent motions, but can be overly conservative. For example, fast rotations in a first-person view may not be handled well, where a great portion of the warped depth buffer can be considered holes. A sophisticated solution to handle this issue could be the composition of the pre-rendering of the previously visible objects with the warped depth buffer, but it would require two-stage geometry passes. We instead propose an efficient workaround, which fills the holes using the motion vectors at the extrema of the local search bound (approximate WOC; AWOC). The source pixels would not converge with their motions, but we simply consider the nearest of them as an approximate depth. Table 7 shows the results of our experiments. As motions increase, AWOC maintains good culling performance, while WOC's performance is reduced to some extent. AWOC can exhibit small errors with respect to WOC in the resulting rendering, but the difference is small (AWOC never descends below 30 dB in terms of PSNR). Also, the errors are hardly perceivable, due to the rapid motion.

6 APPLICATION: SOFT SHADOWS

Our second example application is high-quality soft shadows. These are computed via depth maps generated for each area-light sample. Usually, the many shadow maps would result in high cost, as the rendering has to be repeated as many times as there are light samples. Existing efficient algorithms, such as percentage-closer filtering [Fernando 2005], variance soft shadow mapping [Yang et al. 2010], exponential soft shadow mapping [Shen et al. 2013], and moment shadow mapping [Peters and Klein 2015], do not address true visibility tests, and view-sample methods [Eisemann and Décoret 2007; Sintorn et al. 2008] require more costly specialized geometry processing to avoid multiple rendering passes.

We rely on our warping technique to efficiently produce hundreds of depth maps. Fig. 15 demonstrates examples using our technique compared to reference renderings, varying the number of area light samples (S). We used three depth layers ($L = 3$; one for the ground) and 4 search samples ($N = 4$) for all examples. The generated shadow maps are close to the reference, which enables a correct visibility evaluation without artifacts. Increasing the number of samples, the quality rapidly improves as well (SSIMs and PSNRs up to 1.0 and 46 dB); note that we only used $N = 4$. A small difference only occurs at some depth edges.

The more complex a scene is, the higher the gain of our solution. Table 8 summarizes the performance with respect to a reference rendering. The reference rendering scales linearly with the number of light samples, while ours scales more with resolution due to its image-based nature. In most cases, our solution runs faster, where speed-up factors reach up to 3.4, 5.9, 17.4, and 31.3× for YR, MO, BL, and CB, respectively. Only at 2,048×2,048 resolution in the YR scene, we achieve lower performance, due to the scene's simplicity, which results in geometric rendering being comparable in cost to high-resolution warping.

In practice, when requiring hundreds of views, we can use a coarser but smarter interpolation [Neulander 2008] instead of rendering all the shadow maps, but trade quality for performance.

7 DISCUSSION AND LIMITATIONS

Our depth warping features high performance, stable cost, high scalability, and support for dynamic scenes, which makes our approach interesting for interactive applications.

We proved the effectiveness of our single-view and multi-view depth buffer synthesis. Our unoptimized implementation is still faster than multi-view depth rendering, despite leaving room for further performance improvements. For example, the motion bounds for multiple views can be unified and reused. However, this would lead to a looser bound, which might then require more samples.

Table 8. Performance (frame time measured in ms for the number of area light samples S) comparison of soft shadows rendering with our depth warping and reference geometry rendering. The relative speed-up factors of our solution are measured against the reference using the same number of S.

Scene	Shadow map resolution	Reference (ms)				Depth warping (ms)				Relative speed-up (×)			
		S=4	S=16	S=64	S=256	S=4	S=16	S=64	S=256	S=4	S=16	S=64	S=256
YR	512 × 512	8	25	94	365	6	8	34	108	1.4	3.0	2.8	3.4
	1024 × 1024	8	25	95	372	7	13	37	128	1.2	2.0	2.6	2.9
	2048 × 2048	8	27	100	392	11	30	103	393	0.7	0.9	1.0	1.0
MO	512 × 512	17	55	210	826	11	17	57	140	1.5	3.3	3.7	5.9
	1024 × 1024	17	56	212	841	12	19	60	151	1.4	3.0	3.6	5.6
	2048 × 2048	17	63	221	864	16	34	105	397	1.0	1.8	2.1	2.2
BL	512 × 512	55	181	687	3480	45	51	80	200	1.2	3.5	8.6	17.4
	1024 × 1024	65	232	818	3535	45	52	81	204	1.4	4.5	10.1	17.3
	2048 × 2048	65	232	837	3887	45	52	153	564	1.4	4.4	5.5	6.9
CB	512 × 512	205	663	2380	10021	115	117	209	333	1.8	5.7	11.4	30.1
	1024 × 1024	210	670	2537	10652	117	128	212	341	1.8	5.3	11.9	31.3
	2048 × 2048	213	694	2728	10707	118	128	218	445	1.8	5.4	12.5	24.0

Improving the quality with a unified motion bound for multi-view synthesis is a promising direction for future work.

Our solution exhibits high supra-pixel accuracy, but is limited in sub-pixel accuracy (evidenced by high SSIMs and relatively low PSNRs). This results from the loss of geometric accuracy during rasterization; this is a shared problem of most image-based post-processing. In our case, this is associated with the precision of both the warping and motion vectors. In particular, the errors of the motion vectors reside within a half pixel size, but rapid motion can amplify these errors, when querying the search bound query and impact convergence. This often leads to pixel-size edge differences at complex junctions. A potential improvement can be made using the multisampling of the depth buffers, yet at higher cost.

Another potential source of error is rapid motion (e.g., more than 200 pixels per frame). Such motions cause slower convergence, and might lead to triangles flipping orientation. In the future, we would like to adapt the fixed-point search to take the underlying motion magnitude into account.

A last issue concerns the image periphery. Solutions for the iterative search may lie outside the image boundary. At present, we clamp the search samples, so that they remain in the image area. An accurate handling requires a wider field of view to capture the actual source depths, which leads again to additional costs.

In addition to the occlusion culling and soft-shadow mapping, more potential applications of our depth warping solution include depth-of-field rendering, anti-aliasing (reprojecting multiple locations within pixel), and intra-frame reprojection for virtual reality. In all these cases, having a predictive depth map may help to select the right color samples to reduce errors at reduced cost.

8 CONCLUSION

We have presented a high-quality depth warping technique, which is suitable for GPU-based real-time rendering. The previous fixed-point iteration for color image warping has been made more efficient with our tight bounding method. We further showed that the depth holes can be filled well with deep depth buffers, while still achieving high performance. The number of depth layers can be reduced and bounded with generalized umbra culling. The resulting depth warping has proved its benefit in the two applications. The hierarchy-less screen-space occlusion query leads to a low-latency, stable, and scalable solution. For soft-shadow mapping, the multi-view depth warping proved very beneficial, as it avoids repeated geometry processing.

ACKNOWLEDGMENTS

This work is supported in part by the Global Frontier R&D program through the NRF grant (No. 2018M3A6A3058649), funded by the Korea Government, and the VIDI Grant NextView, funded by the NWO Vernieuwingsimpuls. Correspondence on this article can be addressed to Sungkil Lee.

REFERENCES

Dmitry Andreev. 2010. Real-time frame rate up-conversion for video games: or how to get from 30 to 60 fps for free. In *Proc. ACM SIGGRAPH 2010 Talks*. ACM, 16.

Thaddeus Beier and Shawn Neely. 1992. Feature-based image metamorphosis. *Proc. ACM SIGGRAPH* 26, 2 (1992), 35–42.

Jiří Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. 2004. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum* 23, 3 (2004), 615–624.

Huw Bowles, Kenny Mitchell, Robert W Sumner, Jeremy Moore, and Markus Gross. 2012. Iterative image warping. *Computer graphics forum* 31, 2pt1 (2012), 237–246.

Shenchang Eric Chen and Lance Williams. 1993. View interpolation for image synthesis. In *Proc. ACM SIGGRAPH*. ACM, 279–288.

Adam Cichocki. 2017. Advances in Real-Time Rendering in Games. ACM SIGGRAPH Courses.

R Cunniff, M Craighead, D Ginsburg, K Lefebvre, B Licea-Kane, and N Triantos. 2001. *ARB occlusion query.* Technical Report. NVIDIA and ATI.

X. Décoret. 2005. N-buffers for efficient depth map query. In *Proc. Eurographics*. Wiley Online Library, 393–400.

Piotr Didyk, Elmar Eisemann, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel. 2010a. Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays. *Computer Graphics Forum* 29, 2 (2010), 713–722.

Piotr Didyk, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, and Hans-Peter Seidel. 2010b. Adaptive image-space stereo view synthesis. In *Proc. Vision, Modeling, and Visualization*. 299–306.

Hawar Doghramachi and Jean-Normand Bucci. 2017. Deferred+: Next-Gen Culling and Rendering for Dawn Engine. In *GPU Zen: Advanced Rendering Techniques*, Wolfgang Engel (Ed.). Bowker Identifier Services.

Elmar Eisemann and Xavier Décoret. 2007. Visibility Sampling on GPU and Applications. *Computer Graphics Forum* 26, 3 (2007), 535–544.
Cass Everitt. 2001. Interactive order-independent transparency. *White paper, nVIDIA* 2, 6 (2001), 7.
Randima Fernando. 2005. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*. ACM, 35.
Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. 1996. The lumigraph. In *Proc. ACM SIGGRAPH*. ACM, 43–54.
Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *Proc. ACM SIGGRAPH*. ACM, 171–180.
Michael Guthe, Ákos Balázs, and Reinhard Klein. 2006. Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries. In *Proc. Eurographics Symp. Rendering*. 207–214.
Ulrich Haar and Sebastian Aaltonen. 2015. Advances in Real-Time Rendering in Games. ACM SIGGRAPH Courses.
Yun-Suk Kang and Yo-Sung Ho. 2010. High-quality multi-view depth generation using multiple color and depth cameras. In *Proc. International Conf. Multimedia and Expo*. IEEE, 1405–1410.
Nickolay Kasyan, Nicolas Schulz, and Tiago Sousa. 2011. Secrets of CryENGINE 3 Graphics Technology. ACM SIGGRAPH Courses.
James T Klosowski and Claudio T. Silva. 2000. The prioritized-layered projection algorithm for visible set estimation. *IEEE Trans. Visualization and Computer Graphics* 6, 2 (2000), 108–123.
Christoph Kubisch and Markus Tavenrath. 2014. OpenGL 4.4 Scene Rendering Techniques. GPU Technology Conference.
Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. 2010. Real-Time Lens Blur Effects and Focus Control. *ACM Trans. Graphics* 29, 4 (2010), 65:1–7.
Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. 2008. Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers. *Computer Graphics Forum* 27, 7 (2008), 1955–1962.
Marc Levoy and Pat Hanrahan. 1996. Light field rendering. In *Proc. ACM SIGGRAPH*. ACM, 31–42.
David Luebke and Chris Georges. 1995. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *Proc. ACM Symp. Interactive 3D graphics*. ACM, 105–ff.
William R Mark, Leonard McMillan, and Gary Bishop. 1997. Post-rendering 3D warping. In *Proc. Symp. Interactive 3D Graphics*. ACM, 7–ff.
Oliver Mattausch, Jiří Bittner, A. Jaspe, E. Gobbetti, Michael Wimmer, and Renato Pajarola. 2015. CHC+RT: Coherent Hierarchical Culling for Ray Tracing. *Computer Graphics Forum (Proc. Eurographics)* 34, 2 (2015), 537–548.
Oliver Mattausch, Jiří Bittner, and Michael Wimmer. 2008. CHC++: Coherent hierarchical culling revisited. *Computer Graphics Forum* 27, 2 (2008), 221–230.
Oliver Mattausch, Daniel Scherzer, and Michael Wimmer. 2010. High-Quality Screen-Space Ambient Occlusion using Temporal Coherence. *Computer Graphics Forum* 29, 8 (2010), 2492–2503.
Leonard McMillan and Gary Bishop. 1995. Plenoptic modeling: An image-based rendering system. In *Proc. ACM SIGGRAPH*. ACM, 39–46.
Leonard McMillan Jr. 1997. *An image-based approach to three-dimensional computer graphics*. Ph.D. Dissertation. University of North Carolina Chapel Hill.
Diego Nehab, Pedro V Sander, Jason Lawrence, Natalya Tatarchuk, and John R Isidoro. 2007. Accelerating real-time shading with reverse reprojection caching. In *Proc. Graphics hardware*, Vol. 41. 61–62.
Ivan Neulander. 2008. Pismo: parallax-interpolated shadow map occlusion. ACM SIGGRAPH 2008 talks.
Matthias Nießner and Charles Loop. 2012. Patch-based occlusion culling for hardware tessellation.
Christoph Peters and Reinhard Klein. 2015. Moment shadow mapping. In *Proc. Symp. Interactive 3D Graphics and Games*. ACM, 7–14.
Gilberto Rosado and Rainbow Studios. 2007. Motion Blur as a Post-Processing Effect. In *GPU Gems 3*, Hubert Nguyen (Ed.). Addison-Wesley Professional, 69–117.
Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. 2007. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proc. Eurographics Symp. Rendering*. Eurographics Association, 45–50.
Daniel Scherzer, Michael Schwärzler, Oliver Mattausch, and Michael Wimmer. 2009. Real-time soft shadows using temporal coherence. In *Proc. International Symp. Visual Computing*. Springer, 13–24.
Daniel Scherzer, Lei Yang, Oliver Mattausch, Diego Nehab, Pedro V Sander, Michael Wimmer, and Elmar Eisemann. 2012. Temporal Coherence Methods in Real-Time Rendering. *Computer Graphics Forum* 31, 8 (2012), 2378–2408.
Michael Schwarz and Marc Stamminger. 2008. Quality Scalability of Soft Shadow Mapping. In *Graphics Interface 2008*. 147–154.
W Rrenl Seales, Greg Welch, and Christopher O Jaynes. 1999. Real-time depth warping for 3-d scene reconstruction. In *Proc. IEEE Aerospace Conference*, Vol. 3. IEEE, 413–419.
S. M. Seitz and C. R. Dyer. 1996. View Morphing. In *Proc. ACM SIGGRAPH*. ACM, 21–30.
Li Shen, Jieqing Feng, and Baoguang Yang. 2013. Exponential soft shadow mapping. *Computer graphics forum* 32, 4 (2013), 107–116.
Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. 2008. Sample Based Visibility for Soft Shadows using Alias-free Shadow Maps. *Computer Graphics Forum* 27, 4 (2008), 1285–1292.
Pitchaya Sitthi-amorn, Jason Lawrence, Lei Yang, Pedro V Sander, and Diego Nehab. 2008. An improved shading cache for modern GPUs. In *Proc. Symp. Graphics hardware*. Eurographics Association, 95–101.
Sundar Vedula, Simon Baker, and Takeo Kanade. 2002. Spatio-temporal view interpolation. In *Proc. Rendering Techniques*. 65–76.
Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Trans. Image Proc.* 13, 4 (2004), 600–612.
Lance Williams. 1983. Pyramidal parametrics. *Proc. ACM SIGGRAPH* 17, 3 (1983), 1–11.
Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. 2000. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proc. Eurographics Workshop on Rendering*. 71–82.
Baoguang Yang, Zhao Dong, Jieqing Feng, Hans-Peter Seidel, and Jan Kautz. 2010. Variance soft shadow mapping. *Computer Graphics Forum* 29, 7 (2010), 2127–2134.
Lei Yang, Yu-Chiu Tse, Pedro V Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L Wilkins. 2011. Image-based bidirectional scene reprojection. *ACM Trans. Graphics* 30, 6 (2011), 150:1–10.
Xuan Yu, Rui Wang, and Jingyi Yu. 2010. Real-time Depth of Field Rendering via Dynamic Light Field Generation and Filtering. *Computer Graphics Forum* 29, 7 (2010), 2099–2107.
Sveta Zinger, Luat Do, and PHN de With. 2010. Free-viewpoint depth image based rendering. *Journal of visual communication and image representation* 21, 5 (2010), 533–541.
Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2002. EWA splatting. *IEEE Trans. Visualization and Computer Graphics* 8, 3 (2002), 223–238.

Received Sep. 2017; revised June 2018; accepted July 2018