View-warped Multi-view Soft Shadows for Local Area Lights

Adam Marrs[1,2] Benjamin Watson[2] Christopher Healey[1]

[1]NVIDIA [2]North Carolina State University

Figure 1. Our method's soft shadowing of static and dynamic geometry using an area light. The inlays (right) highlight how our approach has soft penumbra and faithfully represents the occlusion relationship of the character's head and the flagpole. PCSS (persentage closer soft shadows) fails in this situation. HDR-VDP2 heatmaps visualize the perceptual image quality of our algorithm and PCSS compared to a high-quality reference. Blue is high quality and red is low quality.

Abstract

Rendering soft shadows cast by dynamic objects in real time with few visual artifacts is challenging to achieve. We introduce a new algorithm for local light sources that exhibits fewer artifacts than fast single-view approximations and is faster than high-quality multi-view solutions. Inspired by layered depth images, image warping, and point-based rendering, our algorithm traverses complex occluder geometry once and creates an optimized multi-view point cloud as a proxy. We then render many depth maps simultaneously on graphics hardware using GPU Compute. By significantly reducing the time spent producing depth maps, our solution presents a new alternative for applications that cannot yet afford the most accurate methods, but that strive for higher quality shadows than possible with common approximations.

 ISSN 2331-7418

1. Introduction

Shadows provide important perceptual cues about the location, shape, arrangement, and perceived motion of objects and light sources [Kersten et al. 1996]. These cues are even stronger when shadows exhibit natural characteristics, such as penumbra [Mamassian et al. 1998]. Despite much research effort, generating realistic shadows with penumbra *in real time for complex dynamic content* remains an open problem.

The primary task of a soft shadowing algorithm is solving the *visibility factor*, i.e., the proportion of an area light visible to each point on a receiving surface. This problem remains difficult because 1) accurate soft shadowing requires visibility data from *many* light samples across an area light, and 2) *occluder fusion*, the non-trivial interaction of multiple occluding surfaces, prevents the individual (and parallel) processing of visibility data for these many samples [Eisemann et al. 2013]. Recent advances in the precomputation and compression of visibility data solve this problem efficiently for static occluders [Scandolo et al. 2016; Myers 2016]; however, visibility is constantly changing and cannot be precomputed for dynamic occluders and lights.

Multi-view rasterization (MVR) has long been the most reasonable proxy for a ground truth solution that real-time applications can consider to recompute the visibility of dynamic objects every frame. The views rendered with rasterization at area light samples are quite similar and promise computational efficiencies; however, the GPU must traverse and rasterize all occluder geometry for every view rendered, making this approach inefficient for complex dynamic occluder geometry and impractical within real-time budgets on existing GPUs using rasterization APIs. Highly customizable software triangle rasterizers utilizing GPU Compute [Laine and Karras 2011] are an attractive alternative; however, their performance is unable to match hardware-accelerated MVR. We therefore seek an alternative to standard triangle rasterization that exploits today's GPU hardware.

We introduce a new algorithm, called view-warped soft shadows (VWSS), that addresses the MVR bottleneck for dynamic occluders and local area lights. The core of our algorithm is inspired by image warping, which can reduce the computation required to produce similar images by performing geometric transformations on post-render data [Wolberg 1994; McMillan and Bishop 1995; Mark et al. 1997; Walter et al. 1999]. Since views distributed across an area light source are often spatially proximal, the associated depth maps contain highly similar data ideal for reuse. We leverage the discretize-and-reuse strategy of image warping by constructing a point cloud of view-dependent data that is subsequently used to generate a complete set of depth maps. Traversing occluder geometry only once, we rasterize a central view of the area light to produce a multi-view optimized point cloud. We then construct multiple depth maps in each traversal of the points. Inspired by layered depth images [Shade et al. 1998], our algorithm avoids artifacts typically associated with warping

(e.g., at disocclusions) by storing all front-facing rasterized fragments along each ray of a central view enlarged to contain all light sample views.

Our experimentation focuses on local area lights, since these lights benefit the most from the view-dependent sampling and reuse strategy we employ to simplify the soft shadowing workload. The results demonstrate that our method is more accurate than existing fast single-view methods; and faster than existing accurate multi-view solutions. The perceptual image comparison measure HDR-VDP2 [Mantiuk et al. 2011] rates our algorithm's output significantly closer to an offline quality MVR reference than other approximations (see Figure 1), and by reducing the time spent on rendering depth maps, we facilitate a more practical quality-to-performance tradeoff than MVR. We observe that when provided identical point data workloads, notable performance improvements are realized when using GPU Compute instead of the graphics pipeline for parallel point-rendering of depth maps. This makes our approach an ideal candidate for asynchronous execution on newer GPUs and APIs that support this feature. Ultimately, the quality of our algorithm's output is limited by the shadow mapping algorithm from which it is derived; however, these limitations are well understood and typical strategies to mitigate these problems are still effective. We provide example source code of the implementation variations in code listings. We compare our algorithm's performance, visual quality, and temporal stability against other popular methods in motion in the supplementary video.

2. Related Work

Point-based rendering uses points rather than triangles as the primary rendering primitive [Levoy and Whitted 1985]. Complex splatting and interpolation techniques are typically employed to construct complete images from points without discontinuities (also called holes) [Grossman and Dally 1998; Zwicker et al. 2001; Marroquim et al. 2007; Gross and Pfister 2007]. Our approach avoids complicated reconstruction methods by generating frame-specific points in real time tailored to the shadow map resolution and the local area light's multi-view configuration.

Image warping is a discretize-and-reuse strategy designed to improve rendering performance by deriving novel images from previously rendered data [Wolberg 1994]. For example, temporal warping techniques are a critical tool in achieving the low latency and high performance required for virtual reality headsets and stereoscopic displays [McMillan and Bishop 1995; Beeler and Gosalia 2016]. Image warping is most successful when the source and derived images are similar, since the GPU rasterizer emits points with a distribution and sampling density optimized for a single view and output resolution. Consequently, the data necessary to produce a complete novel image is not always available in the source. Previous work addresses this problem by filtering [Walter et al. 1999; Yu et al. 2010] or storing additional data in the source

[Mark et al. 1997; Shade et al. 1998]. Our algorithm stores additional data in our optimized point cloud; this process is discussed further in Section 3.1.

2.1. Shadows

Due to the long history of shadow rendering, we refer the reader to detailed surveys on the topic [Hasenfratz et al. 2003; Eisemann et al. 2011; Scherzer et al. 2011]. In short, accurate methods solve for the visibility factor using either distributed ray tracing [Cook et al. 1984] or by accumulating a large number of rasterized depth maps from multiple views [Haeberli and Akeley 1990], while approximate methods abandon accuracy for plausible—but incorrect—visual results.

Multi-view rasterization (MVR) distributes randomly positioned points across an area light. We refer to these points as *light samples*. Light samples are set as an eye location, and the GPU rasterizes geometry between the eye and the receiver to produce a depth map. To calculate the visibility factor, shadow mapping [Williams 1978] executes for each depth map and the results are averaged [Herf and Heckbert 1996]. The visual quality of this brute-force approach is excellent, since occlusion for each light sample is correctly computed; however, this is often prohibitively slow due to multiple traversals of the geometry. Existing graphics hardware is not designed to render multiple similar views of the same geometry efficiently, performing redundant triangle setup, tessellation, and fragment operations for each view.

Imperfect shadow maps (ISM) approximate visibility for indirect illumination using many low-resolution depth maps, generated by splatting a point-based representation of the geometry [Ritschel et al. 2008]. ISM generates its view-independent point representation either offline or at runtime using GPU tessellation hardware [Barák et al. 2013]. To enable real-time performance, ISM constructs depth maps by splatting each point to one randomly chosen depth map. This contributes to ISM's crude approximation of visibility, which must be hole-filled using a pull-push post-process. Developers rarely use ISM to render shadows cast by direct light sources, since it is unable to capture fine detail and avoid undesirable artifacts. In contrast, our approach uses the rasterizer to generate a set of points specialized to the current area light and restructures computation to allow the use of high-resolution depth maps, with each point splatted to all of them. This denser sampling eliminates the need for any post-processing, while still producing high-quality shadows at practical speeds.

Statistics and filtering-based algorithms approximate soft shadows using visibility data from a single depth map. Although these approaches are fast, they are built upon assumptions that fail in common situations [Eisemann et al. 2011]. Statistics-based methods such as convolution [Annen et al. 2007] and exponential shadow maps [Annen et al. 2008] have fixed-size penumbras and are not capable of reproducing the contact-hardening effect. The percentage closer soft shadows (PCSS) algorithm

varies the size of a percentage closer filter (PCF) kernel [Reeves et al. 1987] to approximate contact hardening [Fernando 2005]. PCSS and methods based on it, such as variance soft shadows [Dong and Yang 2010] and moment soft shadow mapping [Peters et al. 2016], compute an average depth for nearby occluders and use the parallel planes equation to estimate filter-kernel size. This simplification prevents these algorithms from correctly handling occluder fusion when highly differing depths are present at adjacent depth-map texels. Our algorithm makes no approximations of this kind and properly handles occluder fusion. Although statistics and filtering methods are inaccurate, we compare our algorithm to them due to their speed, popularity, and inclusion in recent video-game titles, such as Rockstar's *Grand Theft Auto V* [Burnes 2015b] and Ubisoft's *Assassin's Creed Syndicate* [Burnes 2015a].

3. View-warped Soft Shadows

Our view-warped soft shadows algorithm is a combination of point-generation and rendering approaches that quickly and accurately produce many depth maps for use with multi-view shadow mapping. At a high level, this is accomplished by 1) efficiently transforming complex dynamic occluder geometry from triangles to an optimized multi-view point set, and then 2) rendering multiple depth maps simultaneously by reprojecting, z-buffering, and writing depth results for the point-based geometry proxy in parallel on the GPU. We experimented with two implementations of the VWSS concept: buffered and unbuffered.

Illustrated in Figure 2, the buffered implementation maps directly to the conceptual steps of VWSS. First, the complex occluder geometry is rasterized and the generated fragments are stored as points in an unstructured linear (D3D11 Append/Consume) intermediate buffer. In a second step, depth maps are rendered by dispatching GPU Compute (or Pixel Shader) threads that load point data from the intermediate buffer and reproject, z-buffer, and store depth results for multiple light-sample views to multiple depth maps. We use a single linear intermediate buffer in our implementation, since this data structure simplifies the implementation and maximizes GPU utilization during point rendering. By providing the GPU with one large buffer of points, we ensure the machine is always achieving the full parallelism of which it is capable. For game scenes with large numbers of complex dynamic objects, a single linear intermediate buffer may be insufficient, since not all characters will need to be rendered in a given frame. In this scenario, more advanced data structures or partitioning schemes (e.g., BVH or Octree) are better-tailored to manage this kind of complexity. Regardless of the data structure necessary to optimally traverse and cull the scene data, points should be organized and compacted such that they are fed to the GPU in large chucks that maximize the parallelism afforded by the point representation.

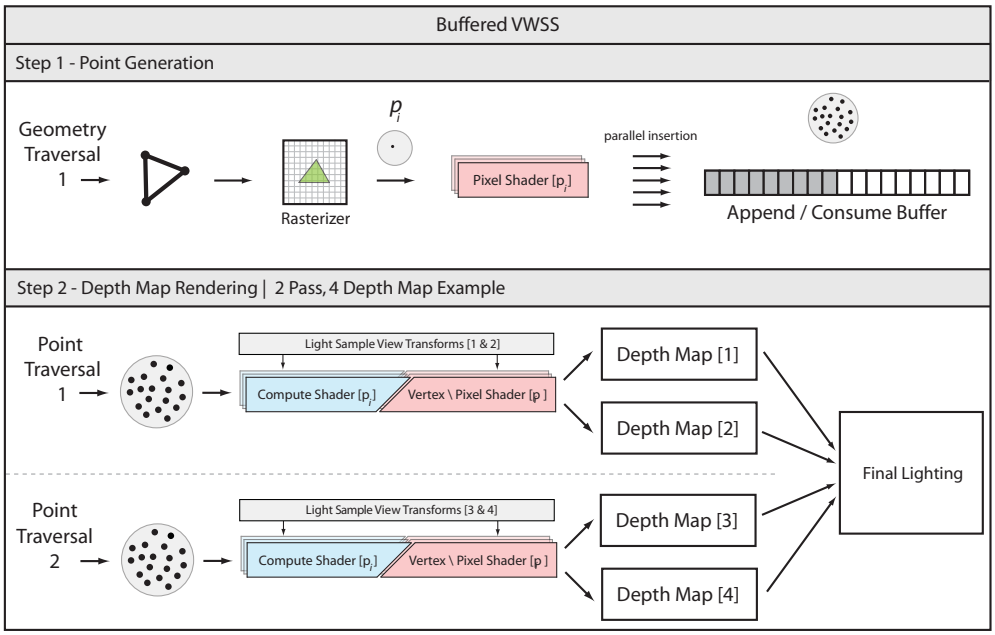

Figure 2. Buffered VWSS. In Step 2, each GPU Compute (or Pixel) Shader thread reprojects, z-buffers, and writes point data for multiple views simultaneously. A single traversal of the point data results in multiple complete depth maps. The number of target depth maps per point traversal and total depth maps is configurable. A two-pass, four depth-map example is shown.

We prefer GPU Compute threads over Pixel Shaders since the compute model provides improved flexibility when scheduling the point-rendering workloads. For even more flexibility, asynchronous GPU Compute may be used to schedule the point-rendering workload; however, our implementation does not yet take advantage of this feature since it is not available in Direct3D 11.1. On platforms that do support it, we suggest inserting asynchronous point-rendering workloads during traditionally low GPU utilization or memory bandwidth tasks, such as shadow depth map rendering for sun-shadow depth-map cascades. Additionally, geometry-heavy tasks where GPU time is dominated by the GPU's fixed-function units (tessellator, rasterizer) are ideal candidates, since asynchronous VWSS workloads will then be able to spread across many more GPU cores without decreasing the coherence of the graphics workload.

Alternatively, our unbuffered implementation combines the point-generation and depth-map rendering steps into a single execution of the graphics pipeline. The point-generation process is identical to the buffered version; however, instead of storing points in an intermediate data structure, they are streamed directly to Pixel Shaders that perform depth-map rendering. This is illustrated in Figure 3. Combining the two steps of VWSS has desirable qualities: 1) we avoid the management of intermediate

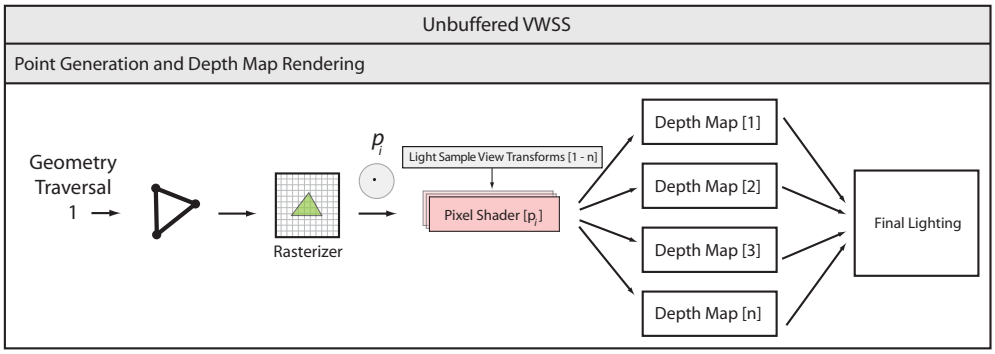

Figure 3. Unbuffered VWSS. Point-generation and depth-map rendering steps are combined into a single execution of the graphics pipeline. Points are streamed from the rasterizer to Pixel Shader threads without storing points in an intermediate buffer.

memory by leveraging the direct connection between the rasterizer and Pixel Shader stages, and 2) we bound total memory usage. These advantages come at a cost though. By combining the two steps of VWSS, the entire algorithm is executed in the GPU graphics pipeline, which decreases the flexibility of workload scheduling and prevents the possibility of point rendering executing as an asynchronous GPU Compute process. Additionally, the parallelism (and thus performance) of unbuffered VWSS is more dependent on GPU memory bandwidth than the buffered approach. This is the case since streaming points from the rasterizer to Pixel Shaders while *simultaneously* writing z-buffered results to several depth maps from Pixel Shaders causes competition for memory bandwidth, decreases coherence, and potentially thrashes the memory subsystem. Consequently, the optimal number of points and target depth maps the unbuffered implementation can handle will be lower than the buffered implementation. Although the performance delta will vary per GPU, the graph at the top of Figure 13 illustrates this tradeoff on a NVIDIA GeForce GTX Titan X (Maxwell).

3.1. Point Generation

VWSS points are generated from triangles by the rasterizer streaming fragments to Pixel Shaders, where the 3D world-space position (XYZ) of each fragment—interpolated as an attribute—is either a) appended to an intermediate buffer (buffered) or b) immediately reprojected, z-buffered, and written to depth maps (unbuffered). We store point world-space positions as three 32-bit floating-point values; however, this can be compressed to half precision to save storage and memory bandwidth. This lower precision option may be acceptable for some use cases, but we prefer the higher-precision format to resolve fine details of character models. During point generation, the rasterizer and output merger do not actually write to a render target; therefore, we avoid allocating memory for the source depth map and instead simply supply the rasterizer with the appropriate viewport dimensions.

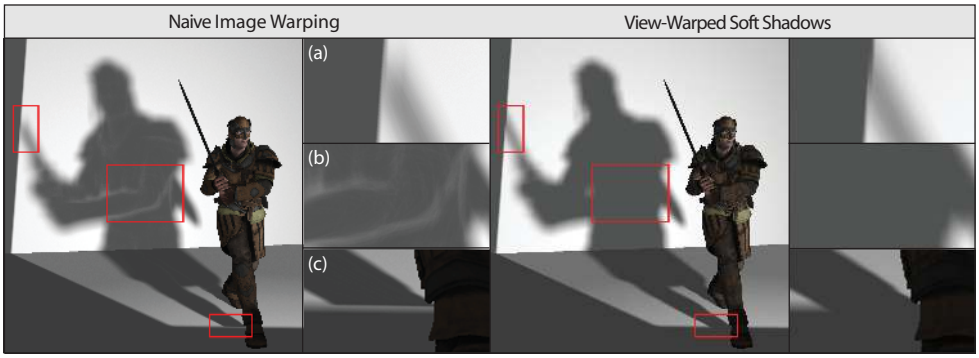

Figure 4. Standard image warping naively applied to multi-view depth map rendering (left) incorrectly leaks light at areas of disocclusion (b) and boundaries of the light sample's field of view (a,c). These common artifacts are eliminated using our approach (right).

Standard image warping reprojects the final result (depth of the closest surface in the case of depth maps) computed from a source view to another nearby target view in space. Naively applying this approach to depth maps when computing multi-view visibility for area lights is insufficient, since the necessary point data to represent all occlusion relationships is not present in the source image. Shown on the left of Figure 4, reprojection of the closest view-dependent samples stored in a typical z-buffer results in incorrect light-leaking artifacts. These artifacts are caused by a lack of data in the source depth map at 1) areas of disocclusion and 2) the boundaries of the area light's influence (imposed by the light sample's field of view).

We solve the lack of data at areas of disocclusion by storing all front-facing samples produced by the rasterizer for the source depth-map buffer (inspired by layered depth images [Shade et al. 1998]). This is illustrated on the right of Figures 4 and 5. Practically, we generate and store these point samples by disabling z-buffering and

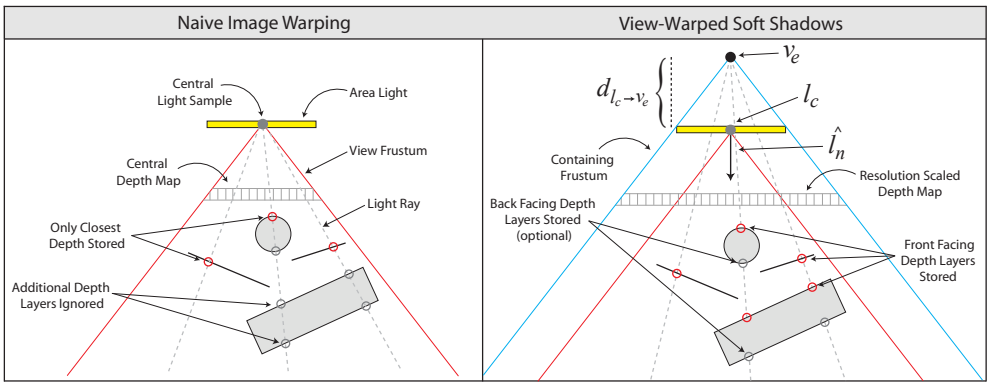

Figure 5. Diagrams of the area light depth-map rendering configuration for naive image warping (left) and our multi-view optimized approach (right).

enabling back-face culling during the rasterization pass that transforms geometry to point samples. This decreases the depth complexity encountered by each light source ray and reduces the points generated. To increase accuracy, back-face culling can be disabled to also capture the back-facing fragments along each light ray. Despite a doubling of the point count, we find the improvement in depth-map rendering accuracy not worth the additional cost of storing and rendering these points. Depth buffering is only performed for complex dynamic and static objects within the local area light's influence. Including all static geometry from a global light source (e.g., the sun) may produce an unwieldy number of points that overwhelms the achievable parallelism of the target GPU. We find that 500,000 to 1.5M points works well on a NVIDIA GeForce GTX Titan X (Maxwell) from 2015.

To address the lack of data at the edges of the light's influence, we transform the source depth map's view to contain the entire area light (as opposed to being placed *on* the area light). See the right diagram of Figure 5. This guarantees that the source depth map includes data relevant to *all* light samples distributed across the area light. In order to maintain a similar sampling rate of each surface compared to MVR, we match this transformation with an increase to the source depth-map's resolution proportional to the distance the source depth-map's view is translated.

To compute the containing source depth-map view, we initially locate the eye $\boldsymbol{v_e}$ at the center of the area light $\boldsymbol{l_c}$. We ensure the eye's vertical field of view fov_v and horizontal field of view fov_h match the respective field(s) of view of the target light sample's depth-map views. Ideally, the field of view aspect ratio of the source and target depth-map views mimic the shape of the area light. We then translate $\boldsymbol{v_e}$ along the negated light normal vector $\widehat{\boldsymbol{l_n}}$ by the distance $d_{l_c \to v_e}$, calculated using Equations (1) and (2), where l_w and l_h are the width and height of a rectangular area light.

$$d_{l_c \to v_e} = \frac{\max(l_w, l_h)}{2 \cdot \tan(\frac{\max(\mathrm{fov_v}, \mathrm{fov_h})}{2})} \tag{1}$$

$$v_e = l_c - (d_{l_c \to v_e} \cdot \widehat{l_n}) \tag{2}$$

$$v_{\mathrm{res}} = \left[m \quad \cdot \quad \frac{z_n + d_{l_c \to v_e}}{z_n} \right]^2 \tag{3}$$

Given a source depth map of resolution m^2, the increased source depth-map resolution v_{res} is computed using Equation (3), where z_n is the source depth-map view's frustum near plane. We locate z_n of the new source depth-map view in the same plane as the area light to avoid rasterizing samples *behind* the area light source. Example C++ code that implements this process using Equations (1), (2), and (3) is provided in Listing 1.

```
// Information about an area light source.
struct LightInfo
{
    float width, height;
    float fov_v, fov_h,
    float nearZ;
};

// Information about a depth map
struct DepthMapInfo
{
    float width, height;
};

/**
* Transform geometry into points tailored for multiple depth maps.
*/
void RenderPoints( DepthMapInfo source, LightInfo light, bool bCullBackFace )
{
    // disable z-buffering (no depth compare) and disable blending
    d3d11Context->OMSetDepthStencilState( DS_NoDepthNoStencil, 0 );
    d3d11Context->OMSetBlendState( BS_None, NULL, D3D11_DEFAULT_SAMPLE_MASK );

    // enable or disable back-face culling
    if( bCullBackFace ) d3d11Context->RSSetState( RS_CullBackFace );
    else d3d11Context->RSSetState( RS_CullNone );

    // transform the source depth map view to contain the area light source
    float HalfMaxFov = max(light.fov_v, light.fov_h) / 2.f;
    float D_LcVe = max(light.width, light.height) / (2.f * tanf(HalfMaxFov));

    // compute the source depth map's viewport resolution
    float Scaling = ( ( light.nearZ + D_LcVe ) / light.nearZ );

    D3D11_VIEWPORT Viewport;
    Viewport.Width = source.width * Scaling;
    Viewport.Height = source.height * Scaling;
    d3d11Context->RSSetViewports( 1, &Viewport );

    //...bind input layout, constant buffers, and shaders
    //...bind append buffer UAV for point storage
    //...draw calls to rasterize occluder geometry
}
```

Listing 1. Example C++ code implementing our multi-view point generation algorithm.

3.2. Depth-Map Rendering (View-Warping)

We render depth maps corresponding to each light sample by reprojecting the points produced by the rasterizer. Both the GPU graphics pipeline and GPU Compute are capable of exploiting locality and increasing parallelism by projecting points into multiple depth buffers during a single thread execution. We classify and compare point-rendering approaches based on the amount of parallelism achieved by each traversal

of the point data (see Figure 13 for performance results). Rendering depth maps from points is accomplished by either producing 1) one depth map per traversal *(serial)*, 2) multiple depth maps per traversal *(partially parallel)*, or 3) all depth maps in a single traversal *(fully parallel)*. The implementation that is most appropriate (graphics or GPU Compute) is dictated by GPU memory bandwidth and the availability of asynchronous compute features. Unlike triangle rasterization, it is possible for either of our implementations to efficiently construct n depth maps with less than n traversals of our point data (illustrated in Figure 2).

We invoke a GPU thread (GPU Compute or Pixel Shader) for each point. In each thread, we read a point's world-space location, apply the view-projection matrix corresponding to each light sample, snap the projected location to the nearest neighbor pixel, and perform z-buffering using atomic functions. Although we are implicitly limiting a point-sample's area of influence to a single depth-map texel, the adjustments to sampling made during point generation allows us to sidestep more complex surface-splatting techniques [Zwicker et al. 2001] and still produce a hole-free result.

The depth maps generated by our algorithm create a very slight bloating of shadow results when compared to the multi-view rasterization reference. The bloating artifact is a result of the differences in how texel coverage is approximated in polygonal rasterization and point-based rendering. Figure 6 illustrates these differences and compares (a) standard polygon rasterization, (b) point-based rendering, and (c) conservative polygon rasterization. All of these approaches are coarse approximations of the actual amount a texel is covered by a polygon, and as such no approach is definitively correct. Standard rasterization is biased towards undersampling, since only texels whose central point is contained within the polygon are considered covered. Conservative rasterization is biased towards oversampling, since all texels intersecting a polygon are marked as covered. Point-based rendering using nearest-neighbor reconstruction presents a middle ground between the two polygon-rasterization options.

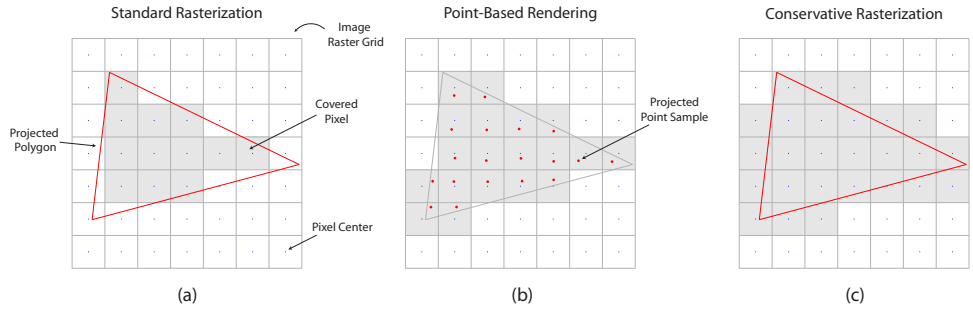

Figure 6. A comparison of texel-coverage approximations. (a) standard polygon rasterization, (b) point-based rendering, and (c) conservative polygon rasterization.

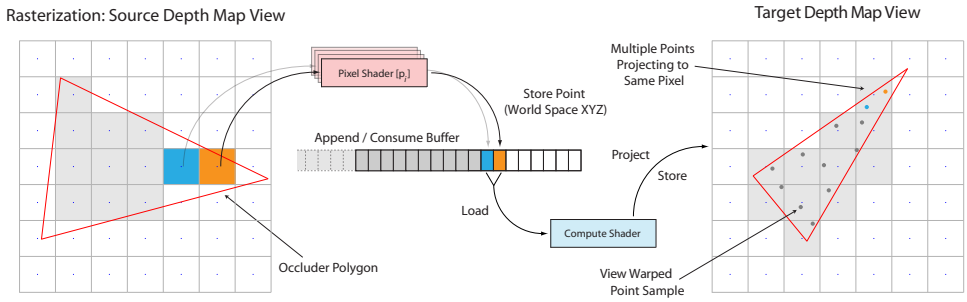

Figure 7. A visualization of buffered VWSS. Note how GPU threads (Compute Shaders in this illustration) may project multiple points to the same destination depth-map texel. Atomic min/max functions are used to resolve these conflicts.

Our optimized point data is often mildly oversampled compared to standard rasterization and causes the visual bloat. If our algorithm's shadows were compared to MVR using conservative rasterization for depth-map rendering, VWSS shadows would instead appear slightly smaller. As the resolution of the depth maps increase the magnitude of the bloat decreases, since the influence of an individual depth texel declines and the various texel coverage approaches converge.

Depending on how GPU memory cache lines are configured, it may be best to load several points in each GPU thread. Also, the change in geometry orientation and occlusion may cause reprojected point samples to project to the same depth-map texel during point rendering. This type of collision is illustrated in Figure 7 and is resolved by atomic min/max (InterlockedMin/Max in D3D11) functions available to Compute and Pixel Shaders. These atomic functions have varying performance characteristics across hardware vendors; however, heavy use of atomics will decrease performance. AMD's GCN architecture exposes instructions for hardware-accelerated 32-bit floating-point min/max atomics on the Sony PlayStation 4® / Microsoft Xbox One® generation of game consoles.

Example depth-map point-rendering Compute Shader HLSL source code is included in Listing 2. Note that atomics functions are performed using 32-bit integer resources since most graphics hardware does not yet support full precision floating-point atomic functions.

```
// ---[ Defines ] ---
#define LIGHT_SAMPLES 32
#define LIGHT_SAMPLES_PER_THREAD 8

// ---[ Resources ] ---
StructuredBuffer<Float3Point>   points          : register( t0 );
RWTexture2DArray<uint>          rt_depth_int    : register( u2 );

// ---[ Constant Buffers ] ---
cbuffer PointRenderingCB                        : register( b12 ) {
    float2 resolution;
    float numPoints;
    float lightSampleOffset;
    matrix viewToLight[LIGHT_SAMPLES];
};

/**
* Warp points, storing depth to an array of depth maps.
* Atomic functions perform z-buffering. Can be used iteratively,
* where the number of depth maps (samples) per thread is specified.
*/
[numthreads( 128, 1, 1 )]
void CS( uint3 threadID : SV_DispatchThreadID ) {

    uint    lightIndex;
    uint    lightSampleMax;
    float   x, y;
    float4  wp, cp;

    // early exit check
    if( threadID.x < ( uint ) numPoints ) {

        wp = float4( points[threadID.x].pos, 1.f );
        lightSampleMax = lightSampleOffset + LIGHT_SAMPLES_PER_THREAD;

        [unroll( LIGHT_SAMPLES_PER_THREAD )]
        for( lightIndex = lightSampleOffset;
                lightIndex < lightSampleMax;
                lightIndex++ )
        {
            // project fragment into view-texture space
            cp = mul( wp, viewToLight[lightIndex] );

            // perspective divide
            cp.xyz /= cp.w;

            // nearest neighbor destination pixel
            x = ( cp.x * resolution.x );
            y = ( cp.y * resolution.y );

            // write to destination with atomic z-buffering
            InterlockedMax( rt_depth_int[ float3( x, y, lightIndex ) ],
                            asuint( cp.z ) );
        }
    }
}
```

Listing 2. Example depth-map point-rendering Compute Shader HLSL source code.

4. Results

We implement our algorithm in a deferred renderer, because these are popular in current commercial game engines (e.g., Unreal® Engine 4, Unity, and Frostbite). The final shaded and shadowed output is constructed by: 1) rendering a G-Buffer (with albedo, normals, and world positions), 2) executing VWSS, MVR, or standard depth-map rendering for PCSS, and 3) executing a fullscreen pass to perform shading and shadow mapping using the depth map(s) produced by each shadowing approach.

Our experimental test environment uses Microsoft Windows 10, Microsoft Direct3D 11.1, Intel i7-4790k CPU @ 4.0 GHz, and a NVIDIA GeForce GTX Titan X (Maxwell) GPU. MVR and VWSS use identical area light-sample positions with Poisson distribution. All depth maps use 32-bit floating-point precision and light sample depth-maps are 1024^2 resolution. Depth-map precision and resolution were chosen based on the needs of our scene and the desire for the highest quality. Half-precision depth maps and alternate resolutions were not tested, but they may provide improved performance in scenarios where the quality tradeoff is acceptable. Our tests use varying polygonal complexity and focus on dynamic (skinned and animated) assets. We have adapted content from Unity Technologies' real-time *Blacksmith* GDC demonstration [Unity 2015]. The geometric complexity of each of our four primary tests are listed in Table 1, and images of them can be found in Figures 1, 4, 8, and 9. We chose a PCSS implementation with a 64-sample blocker search and 96-sample PCF filter, since this is similar to the 'Ultra' preset configuration of PCSS available in some high-end shipping PC game titles. We report GPU times averaged from 1,000 frames of execution, and all VWSS performance numbers are captured from the buffered implementation unless otherwise specified.

We compare the visual quality of all algorithms against a reference image created using an ultra-high quality MVR with 512 rasterized depth maps and a 5-sample PCF filter per pixel (resulting in 2,560 depth samples per pixel). As a numerical measure of image quality, we compute root mean squared error (RMSE). To evaluate perceptual image quality, we use the image comparison measure HDR-VDP2 [Mantiuk et al. 2011], which includes various filters modeling human vision.

4.1. Quality and Performance

Figures 8–13 and Tables 1 and 2 include image-quality comparisons and performance results. Each figure, table, and graph includes detailed captions describing their meaning and a brief analysis of the result shown. A summary analysis of the results presented in the figures, tables, and graphs is included in the discussion in Section 5.

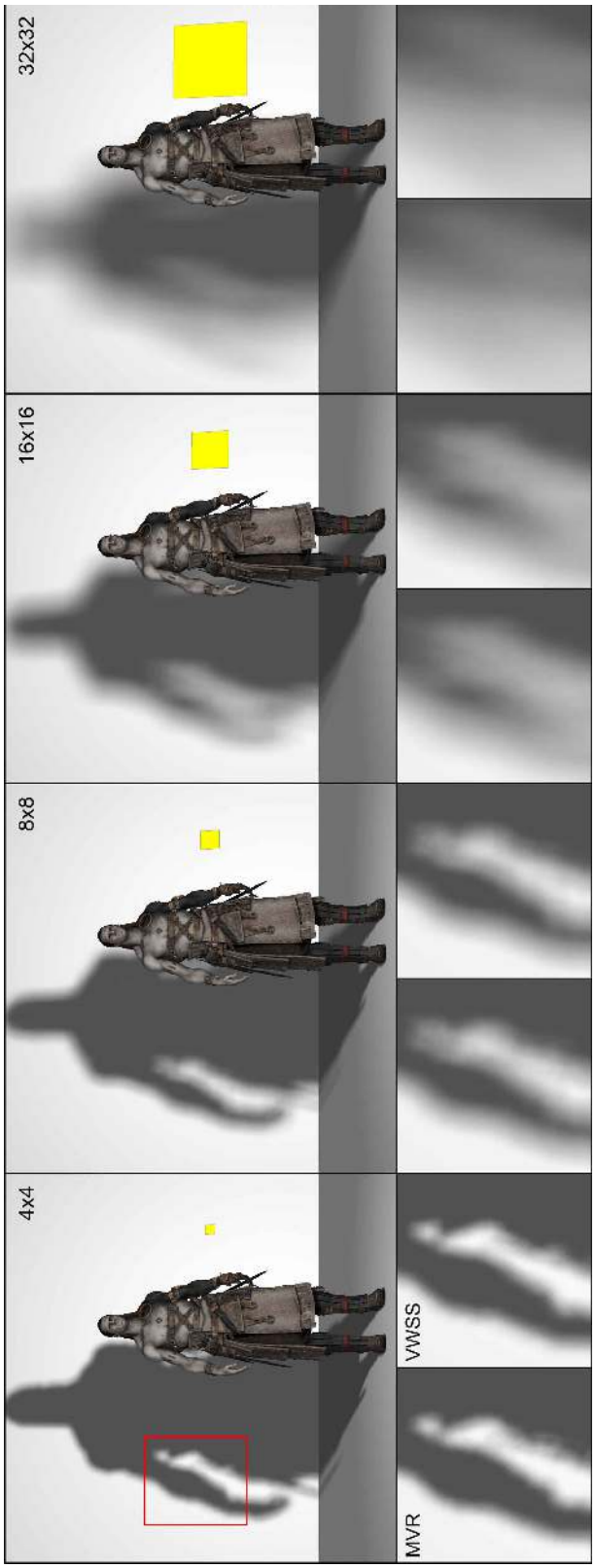

Figure 8. Soft shadowing of the animated Volund character, and a quality comparison of VWSS and MVR as the area light (yellow) grows to 64× its original area. Additional light samples are distributed across the area light as it grows. A constant rate of one light sample per two units of light area is maintained (ranging from 8 to 512 samples). The value of an algorithm based on reprojection is demonstrated by examining how rapidly the quality of the result declines as the synthesized view(s) diverge from the source view. A close-up comparison (bottom) of shadows generated by MVR (left) and VWSS (right) reveals very similar results even as new light samples are further from the source view located at the center of the area light.

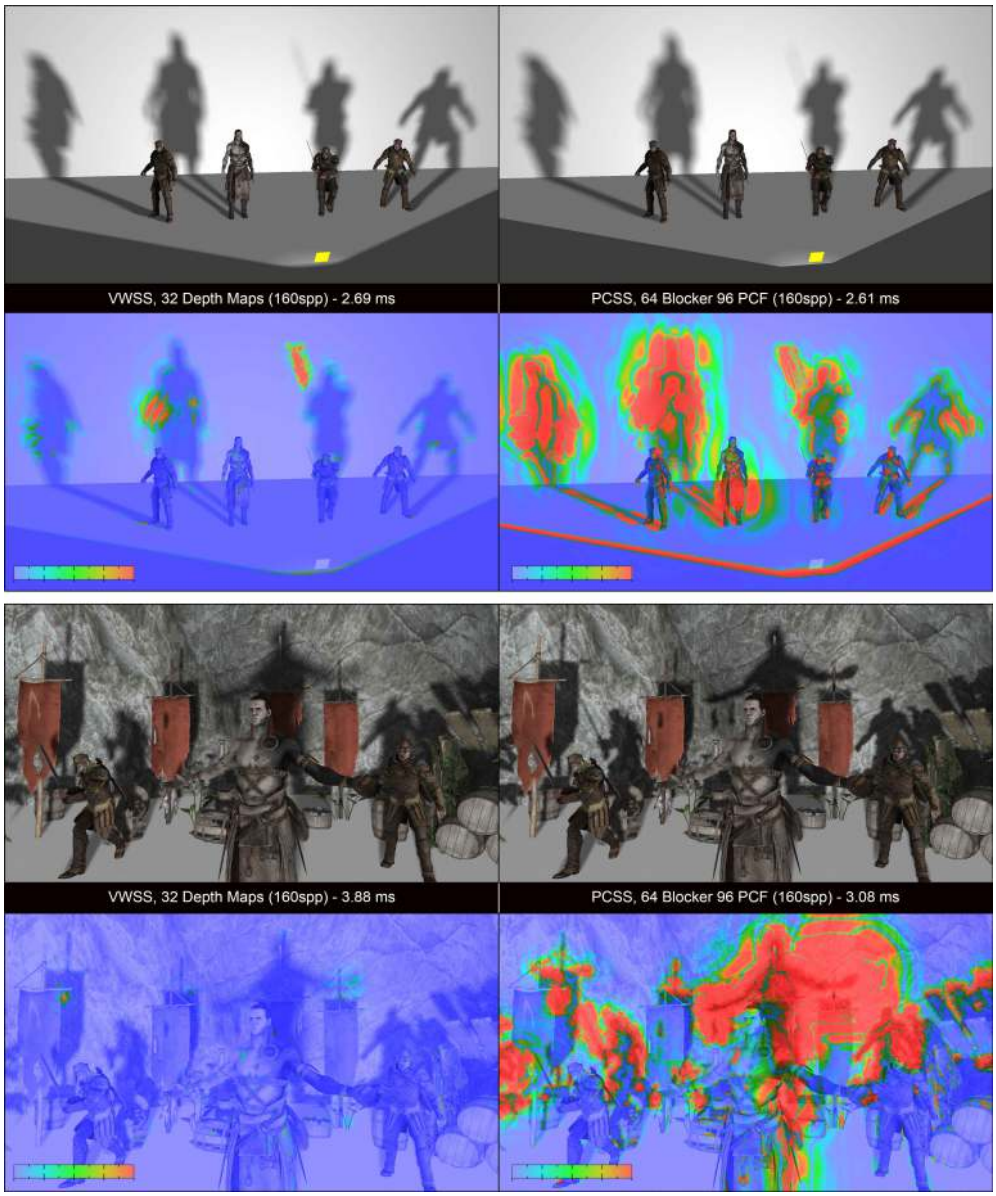

Figure 9. The output of VWSS (left) and PCSS (right) for the Warriors(top) and Cliff-side(bottom) scenes. Perceptual image-quality measure HDR-VDP2 generates heatmaps by comparing each algorithm's output with the high-quality reference image while applying filters that model human vision. Red areas indicate a high probability that differences will be perceived, and blue areas indicate low probability. In these scenes, VWSS renders shadows with higher perceived quality in a similar amount of time as PCSS.

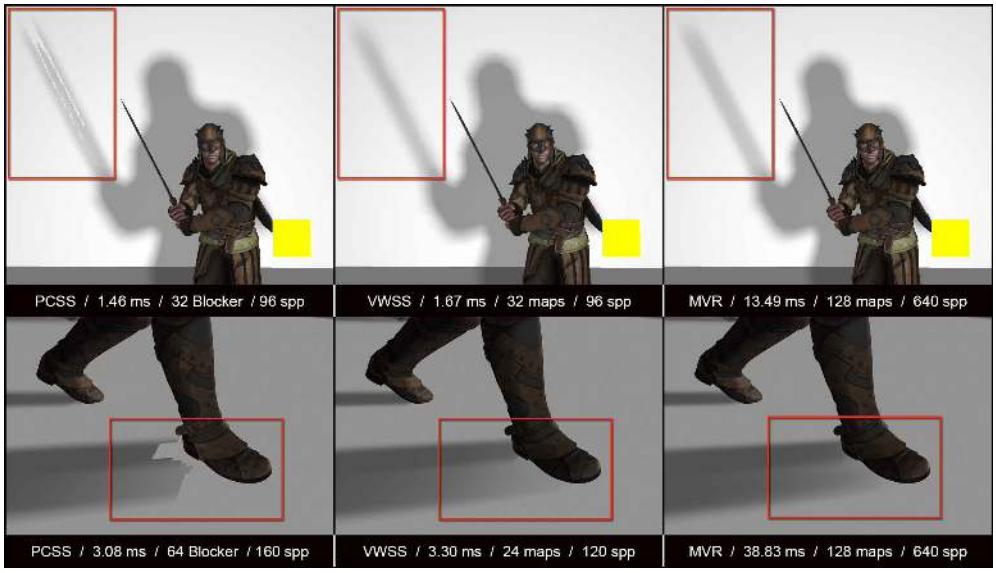

Figure 10. The output of PCSS (left), VWSS (middle), and MVR (right) for the Challenger model. PCSS and other single-view approximations have common failure cases. For example, thin projected geometry that falls between depth-map texels won't be found during the PCSS blocker search step. This is demonstrated by noticeable holes appearing in the sword's shadow (top left). Using an identical number of samples per pixel, VWSS (top middle) has no such failure, delivers quality closer to MVR (top right), and retains rendering time similar to PCSS. Additionally, PCSS-style algorithms that perform a blocker search compute an average of the depth values found. This approximation prevents them from properly handling occluder fusion in some scenarios (bottom left). By z-buffering points projected to many depth maps, VWSS handles occluder fusion properly and produces a result similar to MVR.

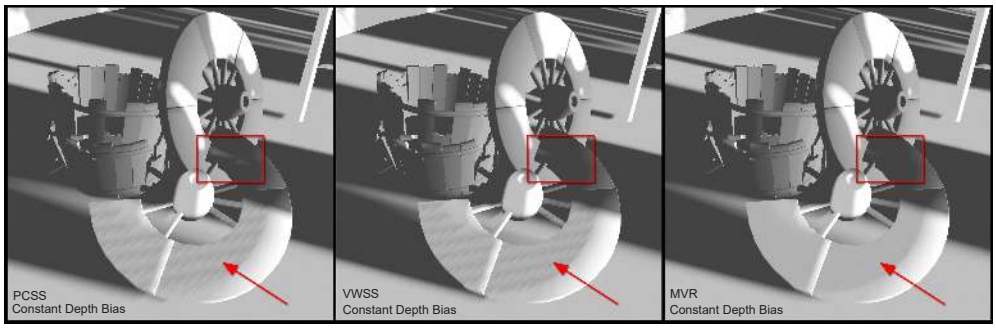

Figure 11. VWSS exhibits self-shadow banding artifacts similar to PCSS when using a constant depth-bias value during shadow mapping. This similarity exists since VWSS still samples occluder geometry from only a single view. MVR does not have these banding artifacts since it rasterizes (resamples) polygons for every depth map before applying a depth bias. Despite the banding, VWSS introduces less incorrect light leaking than PCSS in complex occlusion scenarios (box highlight). Images modified to emphasize self shadowing.

GPU Performance of Soft-Shadowing Algorithms on Various Scenes

Multi-View Algorithms use 24 Depth Maps @ 120 spp

Scene	# of Triangles	# of Points	Algorithm	G-Buffer	Point Generation	Depth Maps	Lighting	Total	SD	×Faster Depth	×Faster Total	RMSE
Challenger	90,007	40,541	MVR	0.261		0.860	1.300	2.421	0.161			1.497
			VWSS	0.262	0.115	0.286	1.259	**1.922**	0.421	2.15	1.26	**1.801**
			PCSS	0.262		0.021	1.590	**1.873**	0.092			**2.753**
Volund	240,186	57,188	MVR	0.300		1.358	1.308	2.966	0.149			1.995
			VWSS	0.293	0.109	0.261	1.272	**1.935**	0.131	3.67	1.53	**1.428**
			PCSS	0.291		0.052	1.767	**2.110**	0.105			**2.503**
Warriors	510,523	101,002	MVR	0.400		4.095	1.280	5.775	0.298			1.437
			VWSS	0.397	0.471	0.342	1.120	**2.330**	0.481	5.04	2.48	**1.627**
			PCSS	0.394		0.157	2.060	**2.611**	0.144			**3.608**
Cliffside	672,927	347,060	MVR	0.683		5.148	1.271	7.102	0.540			1.129
			VWSS	0.660	0.746	0.636	1.261	**3.303**	0.575	3.73	2.15	**1.316**
			PCSS	0.681		0.153	2.246	**3.080**	0.161			**3.232**

Table 1. GeForce GTX Titan X (Maxwell) GPU performance of VWSS compared to MVR and PCSS. VWSS and MVR use 24 depth maps of 1024^2 resolution and 32-bit precision. Each screen pixel uses a 5 sample PCF filer on each depth map, resulting in 120 depth-map samples per pixel. PCSS uses a single depth map, a 64 sample blocker search, and a 96 sample PCF filter, resulting in 160 depth-map samples per pixel. VWSS accelerates multi-view depth-map rendering time ∼2×-5× (highlighted blue) and improves on MVR's total rendering time up to 2.5× (highlighted red). Although VWSS uses fewer total visibility samples, it matches PCSS performance (bolded) while producing images with less total error (RMSE). Scenes with complex geometry benefit the most from VWSS. GPU times reported in milliseconds.

GPU Performance of Soft-Shadowing Algorithms on Various Scenes

Multi-View Algorithms use 128 Depth Maps @ 640 spp

Scene	# of Triangles	# of Points	Algorithm	G-Buffer	Point Generation	Depth Maps	Lighting	Total	SD	×Faster Depth	×Faster Total	RMSE
Challenger	90,007	40,541	MVR	0.261		5.396	7.822	13.485	1.709			0.407
			VWSS	0.262	0.115	1.118	7.757	9.260	0.943	4.38	1.46	**0.774**
			PCSS	0.262		0.021	1.590	1.873	0.092			**2.753**
Volund	240,186	57,188	MVR	0.300		10.816	8.242	19.371	2.928			0.411
			VWSS	0.293	0.109	1.216	7.952	9.588	0.795	8.16	2.02	**0.772**
			PCSS	0.291		0.052	1.767	2.110	0.105			**2.503**
Warriors	510,523	101,002	MVR	0.400		22.101	7.469	29.990	5.876			0.403
			VWSS	0.397	0.471	1.253	6.938	9.098	0.440	12.82	3.30	**0.702**
			PCSS	0.394		0.157	2.060	2.611	0.144			**3.608**
Cliffside	672,927	347,060	MVR	0.683		30.320	7.679	38.834	6.243			0.255
			VWSS	0.660	0.746	2.910	6.823	11.525	0.821	8.29	3.37	**1.129**
			PCSS	0.681		0.153	2.246	3.080	0.161			**3.232**

Table 2. GeForce GTX Titan X (Maxwell) GPU performance of VWSS compared to MVR and PCSS. VWSS and MVR use 128 depth maps of 1024^2 resolution and 32-bit precision. Each screen pixel uses a 5 sample PCF filer on each depth map, resulting in 640 depth-map samples per pixel. PCSS uses a single depth map, a 64 sample blocker search, and a 96 sample PCF filter, resulting in 160 depth-map samples per pixel. VWSS accelerates multi-view depth-map rendering time ∼4×-13× (highlighted blue) and improves on MVR's total rendering time up to 3× (highlighted red). Despite being slower than PCSS, the increased light-sample counts of VWSS significantly improves RMSE at a lower cost than MVR. Scenes with complex geometry benefit the most from VWSS. GPU times reported in milliseconds.

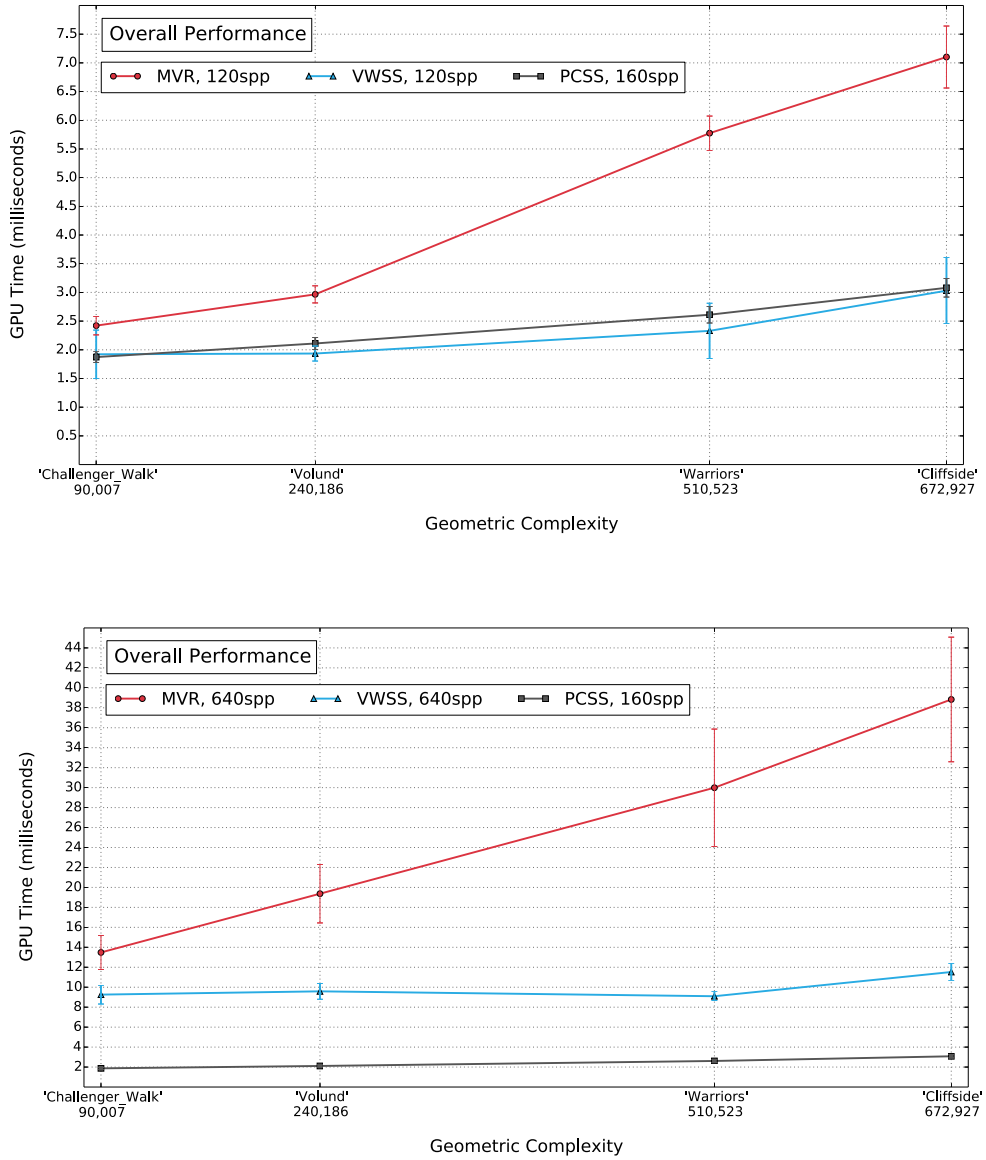

Figure 12. GeForce GTX Titan X (Maxwell) GPU performance of shadowing algorithms as geometric complexity increases. The top graph plots the data presented in Table 1, and the bottom graph plots data from Table 2. Multi-view rasterization (red) exhibits the expected linear rise in rendering time as geometric complexity increases. Unlike MVR, the performance of VWSS (blue) is weakly linked to the scene geometry. Instead, VWSS performance is influenced by the number of points and target depth maps. VWSS performance is also more stable than MVR, as shown by the lower standard deviation (especially at higher geometry complexity and depth-map counts).

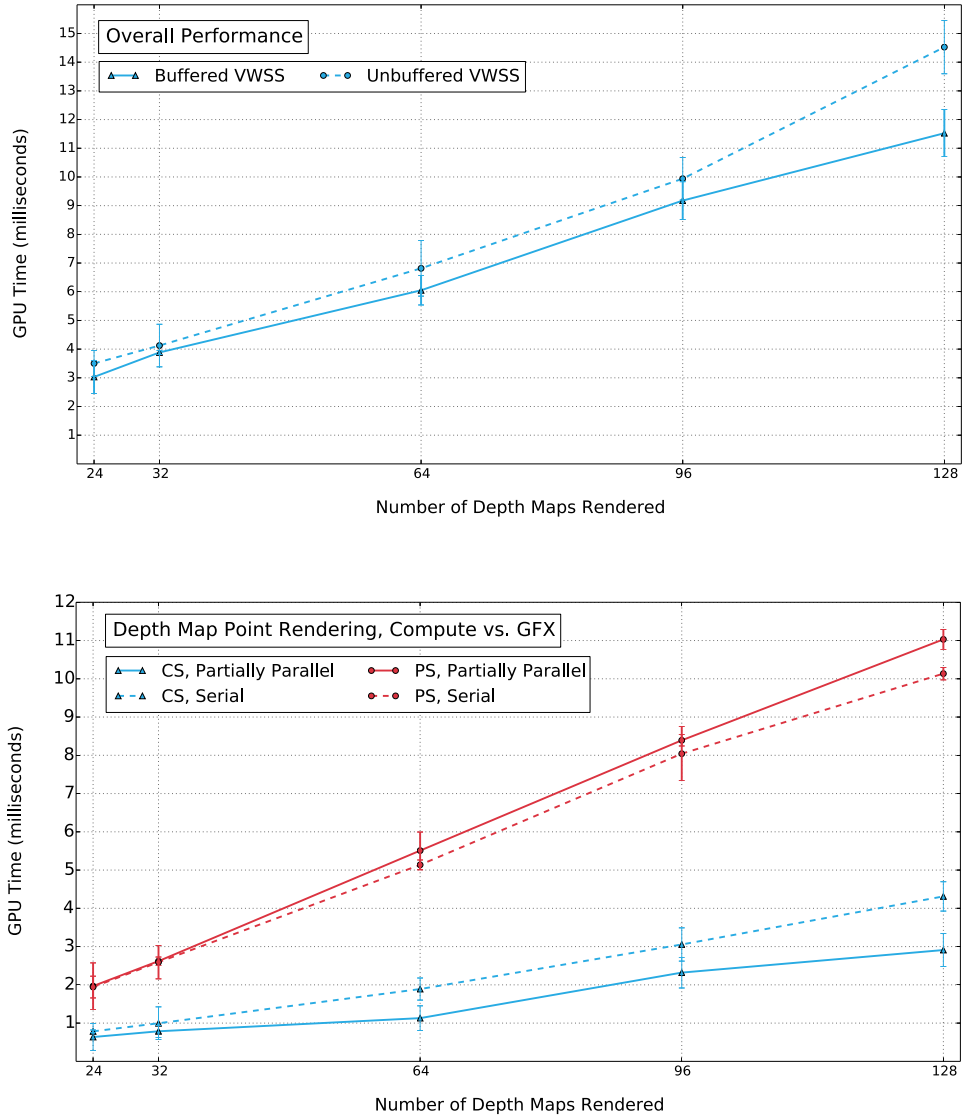

Figure 13. Top: A comparison of GPU performance of the buffered and unbuffered VWSS implementations described Section 3, with an increasing number of depth maps. As the number of depth maps increases, the unbuffered implementation incurs a performance penalty due to memory-bandwidth saturation during its fully parallel execution. Bottom: A comparison of point-rendering implementations using GPU Compute (blue) or the programmable GPU graphics pipeline (red). Each implementation processes an identical workload. The GPU graphics pipeline is unable to render points in a partially parallel manner without decreasing performance. Unexpectedly, a serial GPU graphics pipeline implementation is slower than serial GPU compute, especially as the number of rendered depth maps increases. This demonstrates that passing point data through the GPU graphics pipeline as vertices rather than directly loading them in GPU Compute, or as output generated from rasterized triangles, reduces the achievable performance.

5. Discussion

Our algorithm establishes a useful new point on the quality-to-performance continuum for real-time applications rendering soft shadows using local area lights and complex dynamic occluders. Scenes with high-complexity occluders benefit the most. Performance is faster than MVR since we traverse occluder geometry only once, improve parallelism with an optimized point representation, and exploit locality while rendering depth maps. As the geometric complexity of the scene decreases, the bottleneck we address no longer exists and brute-force rasterization is acceptable.

Although our approach has similarities to imperfect shadow mapping(ISM), it takes a fundamentally different approach aimed specifically at exploiting current GPU hardware to produce high-quality shadows cast by direct light sources in real time. ISM also uses point clouds, but our approach produces point data specific to the current area light, using rasterization hardware ideal for view-specific real-time point generation. Both algorithms create many shadow maps, but we restructure the rendering computation to increase GPU utilization, enabling a denser sampling of visibility.

Of our two VWSS implementations— buffered and unbuffered— the buffered implementation is faster. Although there are likely scenarios where this relationship isn't true, the unbuffered implementation's single pass execution in the graphics pipeline is usually stalled by memory-bandwidth contention and saturation. Despite this, the advantage of simplified and bounded memory management may be well worth the performance penalty in certain cases. For example, some order-independent transparency algorithms that sort per-pixel fragments [Wyman 2016] can be challenging to use since they must work around unbounded memory requirements.

We experimented with point-rendering implementations using both GPU Compute and the GPU graphics pipeline. GPU Compute is faster in every configuration (serial, partially parallel, fully parallel) when processing identical point workloads. We attribute this performance difference to 1) unnecessary setup and raster operations that the graphics pipeline performs on point primitives and 2) increased parallelism achieved in GPU Compute from the ability to explicitly control scheduling. As a result, our recommendation is to use GPU Compute for point rendering. We are pursuing further gains that may be realized by asynchronous compute on platforms that support it.

We evaluated the geometric and temporal aliasing of our technique compared to MVR and PCSS. Since our method uses multiple depth maps, it exhibits less aliasing of both types compared to PCSS. However, since we do not resample the occluder geometry for each depth map, aliasing is still more pronounced than MVR. We experimented with temporal averaging, inspired by previous temporal improvements to shadows [Scherzer et al. 2009; Schwärzler et al. 2013], to improve VWSS aliasing. The preliminary results are encouraging (see the supplementary video), but can suffer from ghosting artifacts like most temporal methods.

5.1. Limitations

There are scenarios where our point-based approach is not optimal. When only a few occluding polygons exist that generate many points in the light's view, the geometry is easily cached and brute-force rasterization is better designed for this kind of workload. This scenario is becoming less common since the complexity of real-time geometry is increasing faster than image resolution [Gross and Pfister 2007], but pathological situations can still present themselves.

Our solution makes heavy use of GPU storage and memory bandwidth. As a consequence, the lighting phase remains a challenging problem. In the near term, rendering screen-space lighting at a lower resolution, utilizing dynamically varying lower resolution or precision depth maps, employing reconstruction filtering techniques, or adopting temporal and/or stochastic methods may improve lighting performance at varying costs to image quality and latency. Asynchronous compute may provide the opportunity to pipeline and overlap the depth-map rendering and lighting steps, filling utilization gaps. Longer term, better compression techniques may emerge for multi-view depth maps. We do not integrate any of these optimization solutions in our algorithm, since they are not central to our contribution, are applicable to nearly all shadowing algorithms, and no single solution is best in all cases.

Similar to PCSS, our approach displays bands of self-shadowing. Even though we store more information than a typical depth map, occluder geometry is still sampled from a single view. After shadow-map depth biasing, the reprojected (warped) samples we produce are less accurate than those produced by resampling occluder geometry in rasterization. Beyond this, VWSS shares the same core limitations as all algorithms based on shadow mapping. Several decades of refinements and workarounds can be applied to VWSS, just as they would for any shadow-mapping algorithm. For clarity and simplicity of evaluation, we have not employed them here.

5.2. Conclusion

Soft shadows are one of many complex lighting phenomena that can be approximated using multi-view rendering. Unfortunately, today's GPUs do not support multi-view rendering well, even when those views are quite similar to one another. We chose points produced frame-by-frame to avoid the existing limitations; however, current GPU hardware can be improved for point rendering. Specifically, robust support for high-performance atomic functions and increased memory bandwidth are critical to improve multi-view z-buffering in GPU compute.

As useful as they are, points are not a direct attack on the multi-view problem. Utilizing a hardware-accelerated GPU graphics pipeline to render multiple views in a single pass is essential. In the short term, this may be accomplished by removing limitations of the Geometry Shader, the rasterizer, or using ray tracing as its viability matures. In the longer term, the GPU graphics pipeline should be improved to effi-

ciently support multi-view rendering directly from triangles, without a transformation into points. A potential alternative may be to create a separate, more efficient pathway designed for indirect effects by providing a flexible compute system with direct access to hardware-accelerated processing of geometry.

The point generation, warping, and parallelism we exploit might also find application elsewhere. For VWSS, high-quality image warping is enabled by the viewing constraints typical of real-time area lights. Other applications with similar constraints may benefit from our approach. Soft shadows cast from omni-directional and volume light sources, depth of field, and ambient occlusion may all benefit, since they rely heavily on depth maps and their approximations require many views.

Acknowledgements

We thank Unity Technologies for releasing the *Blacksmith* demo content to the public for free. We also thank David Luebke, Josef Spjut, Turner Whitted, and Jon Story for helpful discussions.

References

ANNEN, T., MERTENS, T., BEKAERT, P., SEIDEL, H.-P., AND KAUTZ, J. 2007. Convolution Shadow Maps. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, EGSR'07, 51–60. doi:10.2312/EGWR/EGSR07/051-060. 4

ANNEN, T., MERTENS, T., SEIDEL, H.-P., FLERACKERS, E., AND KAUTZ, J. 2008. Exponential Shadow Maps. In *Proceedings of graphics interface 2008*, Canadian Information Processing Society, Toronto, Ont., Canada, GI '08, 155–161. URL: http://dl.acm.org/citation.cfm?id=1375714.1375741. 4

BARÁK, T., BITTNER, J., AND HAVRAN, V. 2013. Temporally Coherent Adaptive Sampling for Imperfect Shadow Maps. In *Proceedings of the Eurographics Symposium on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, EGSR '13, 87–96. doi:10.1111/cgf.12154. 4

BEELER, D., AND GOSALIA, A., 2016. Asynchronous Timewarp on Oculus Rift. https://developer.oculus.com/blog/asynchronous-timewarp-on-oculus-rift/, March. 3

BURNES, A., 2015. Assassin's Creed Syndicate Graphics and Performance Guide. http://www.geforce.com/whats-new/guides/assassins-creed-syndicate-graphics-and-performance-guide, November. 5

BURNES, A., 2015. Grand Theft Auto V PC Graphics and Performance Guide. https://www.geforce.com/whats-new/guides/grand-theft-auto-v-pc-graphics-and-performance-guide, April. 5

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed Ray Tracing. *SIGGRAPH Comput. Graph. 18*, 3 (Jan.), 137–145. URL: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/964965.808590, doi:10.1145/964965.808590. 4

DONG, Z., AND YANG, B. 2010. Variance Soft Shadow Mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, I3D '10, 18:1–18:1. URL: http://doi.acm.org/10.1145/1730804.1730990, doi:10.1145/1730804.1730990. 5

EISEMANN, E., SCHWARZ, M., ASSARSSON, U., AND WIMMER, M. 2011. *Real-Time Shadows*. A K Peters, Natick, MA. URL: http://www.cg.tuwien.ac.at/research/publications/2011/EISEMANN-2011-RTS/. 4

EISEMANN, E., ASSARSSON, U., SCHWARZ, M., VALIENT, M., AND WIMMER, M. 2013. Efficient Real-time Shadows. In *ACM SIGGRAPH 2013 Courses*, ACM, New York, NY, SIGGRAPH '13, 18:1–18:54. URL: http://doi.acm.org/10.1145/2504435.2504453, doi:10.1145/2504435.2504453. 2

FERNANDO, R. 2005. Percentage-Closer Soft Shadows. In *ACM SIGGRAPH 2005 Sketches*, ACM, New York, NY, SIGGRAPH '05. URL: http://doi.acm.org/10.1145/1187112.1187153, doi:10.1145/1187112.1187153. 5

GROSS, M., AND PFISTER, H. 2007. *Point-Based Graphics*. Morgan Kaufmann Publishers Inc., San Francisco, CA. 3, 23

GROSSMAN, J. P., AND DALLY, W. J. 1998. Point Sample Rendering. In *Rendering Techniques 98*, Springer, Berlin, 181–192. 3

HAEBERLI, P., AND AKELEY, K. 1990. The Accumulation Buffer: Hardware Support for High-quality Rendering. *SIGGRAPH Comput. Graph. 24*, 4 (Sept.), 309–318. URL: http://doi.acm.org/10.1145/97880.97913, doi:10.1145/97880.97913. 4

HASENFRATZ, J. M., LAPIERRE, M., HOLZSCHUCH, N., SILLION, F., AND GRAVIR/IMAG-INRIA, A. 2003. A Survey of Real-time Soft Shadows Algorithms. *Computer Graphics Forum 22*, 4, 753–774. doi:10.1111/j.1467-8659.2003.00722.x. 4

HERF, M., AND HECKBERT, P. S. 1996. Fast Soft Shadows. In *ACM SIGGRAPH 96 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96*, ACM, New York, NY, SIGGRAPH '96, 145–. URL: http://doi.acm.org/10.1145/253607.253870, doi:10.1145/253607.253870. 4

KERSTEN, D., D., K., P., M., AND BLTHOFF, I. 1996. Illusory Motion from Shadows. *Nature 379*, 6560, 31–31. URL: http://www.nature.com/nature/journal/v379/n6560/full/379031a0.html, doi:10.1038/379031a0. 2

LAINE, S., AND KARRAS, T. 2011. High-performance Software Rasterization on GPUs. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, ACM, New York, NY, HPG '11, 79–88. URL: http://doi.acm.org/10.1145/2018323.2018337, doi:10.1145/2018323.2018337. 2

LEVOY, M., AND WHITTED, T. 1985. The Use of Points as a Display Primitive. *Technical report, Computer Science Department, University of North Carolina at Chapel Hill* (Jan.). 3

MAMASSIAN, P., KNILL, D., AND D, K. 1998. The Perception of Cast Shadows. *Trends in Cognitive Sciences 2*, 8 (August), 288–295. 2

MANTIUK, R., KIM, K. J., REMPEL, A. G., AND HEIDRICH, W. 2011. HDR-VDP-2: A Calibrated Visual Metric for Visibility and Quality Predictions in All Luminance Conditions. In *ACM SIGGRAPH 2011 Papers*, ACM, New York, NY, SIGGRAPH '11, 40:1–40:14. URL: http://doi.acm.org/10.1145/1964921.1964935, doi:10.1145/1964921.1964935. 3, 14

MARK, W. R., MCMILLAN, L., AND BISHOP, G. 1997. Post-rendering 3D Warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, ACM, New York, NY, I3D '97, 7–ff. URL: http://doi.acm.org/10.1145/253284.253292, doi:10.1145/253284.253292. 2, 4

MARROQUIM, R., KRAUS, M., AND CAVALCANTI, P. R. 2007. Efficient Point-Based Rendering Using Image Reconstruction. In *Eurographics Symposium on Point-Based Graphics*, The Eurographics Association, Aire-la-Ville, Switzerland, M. Botsch, R. Pajarola, B. Chen, and M. Zwicker, Eds. doi:10.2312/SPBG/SPBG07/101-108. 3

MCMILLAN, L., AND BISHOP, G. 1995. Head-Tracked Stereoscopic Display Using Image Warping. In *Proceedings SPIE, Volume 2409*, SPIE, Bellingham, WA, 21–30. 2, 3

MYERS, K. 2016. Sparse Shadow Tree. In *ACM SIGGRAPH 2016 Talks*, ACM, New York, NY, SIGGRAPH '16, 14:1–14:2. URL: http://doi.acm.org/10.1145/2897839.2927418, doi:10.1145/2897839.2927418. 2

PETERS, C., MUNSTERMANN, C., WETZSTEIN, N., AND KLEIN, R. 2016. Beyond Hard Shadows: Moment Shadow Maps for Single Scattering, Soft Shadows and Translucent Occluders. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, I3D '16, 159–170. URL: http://doi.acm.org/10.1145/2856400.2856402, doi:10.1145/2856400.2856402. 5

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering Antialiased Shadows with Depth Maps. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 283–291. URL: http://doi.acm.org/10.1145/37402.37435, doi:10.1145/37402.37435. 5

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph. 27*, 5 (Dec.), 129:1–129:8. URL: http://doi.acm.org/10.1145/1409060.1409082, doi:10.1145/1409060.1409082. 4

SCANDOLO, L., BAUSZAT, P., AND EISEMANN, E. 2016. Compressed Multiresolution Hierarchies for High-Quality Precomputed Shadows. *Computer Graphics Forum (Proc. Eurographics) 35*, 2 (May). URL: http://graphics.tudelft.nl/Publications-new/2016/SBE16. 2

SCHERZER, D., SCHWÄRZLER, M., MATTAUSCH, O., AND WIMMER, M. 2009. Real-Time Soft Shadows Using Temporal Coherence. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II*, Springer-Verlag, Berlin, Heidelberg, ISVC '09, 13–24. doi:10.1007/978-3-642-10520-3_2. 22

SCHERZER, D., WIMMER, M., AND PURGATHOFER, W. 2011. A Survey of Real-Time Hard Shadow Mapping Methods. *Computer Graphics Forum 30*, 1, 169–186. doi:10.1111/j.1467-8659.2010.01841.x. 4

SCHWÄRZLER, M., LUKSCH, C., SCHERZER, D., AND WIMMER, M. 2013. Fast Percentage Closer Soft Shadows using Temporal Coherence. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, I3D '13, 79–86. URL: http://doi.acm.org/10.1145/2448196.2448209, doi:10.1145/2448196.2448209. 22

SHADE, J., GORTLER, S., HE, L.-W., AND SZELISKI, R. 1998. Layered Depth Images. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, SIGGRAPH '98, ACM, 231–242. URL: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/280814.280882, doi:10.1145/280814.280882. 2, 4, 8

UNITY, 2015. The Blacksmith: Characters. https://assetstore.unity.com/packages/essentials/asset-packs/the-blacksmith-characters-39941 March. 14

WALTER, B., DRETTAKIS, G., AND PARKER, S. 1999. Interactive Rendering Using the Render Cache. In *Proceedings of the 10th Eurographics Conference on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, EGWR'99, 19–30. doi:10.2312/EGWR/EGWR99/019-030. 2, 3

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug.), 270–274. URL: http://doi.acm.org/10.1145/965139.807402, doi:10.1145/965139.807402. 4

WOLBERG, G. 1994. *Digital Image Warping*, 1st ed. IEEE Computer Society Press, Los Alamitos, CA. 2, 3

WYMAN, C. 2016. Exploring and Expanding the Continuum of OIT Algorithms. In *Proceedings of High Performance Graphics*, Eurographics Association, Aire-la-Ville, Switzerland, HPG '16, 1–11. URL: http://dl.acm.org/citation.cfm?id=2977336.2977338. 22

YU, X., WANG, R., AND YU, J. 2010. Real-time Depth of Field Rendering via Dynamic Light Field Generation and Filtering. In *Computer Graphics Forum*, 2099–2107. URL: http://doi.acm.org/10.1145/1730804.1730990, doi:10.1111/j.1467-8659.2010.01797.x. 3

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. Surface Splatting. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, SIGGRAPH '01, 371–378. URL: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/383259.383300, doi:10.1145/383259.383300. 3, 11

Index of Supplemental Materials

A supplementary video includes real-time captures of our algorithm's output, along with direct comparisons with multi-view rasterization and percentage closer soft shadows.

Author Contact Information

Adam Marrs
NVIDIA Corporation
2700 Meridian Parkway
Suite 100
Durham, NC 27713
amarrs@nvidia.com
http://www.visualextract.com

Benjamin Watson
NC State University
2280 Engineering Building 2
890 Oval Drive
Raleigh, NC 27606
bwatson@ncsu.edu

Christopher G. Healey
NC State University
2266 Engineering Building 2
890 Oval Drive
Raleigh, NC 27606
healey@ncsu.edu

Adam Marrs, Benjamin Watson, and Christopher Healey, View-warped Multi-view Soft Shadowing for Local Area Lights, *Journal of Computer Graphics Techniques (JCGT)*, vol. 7, no. 3, 1–28, 2018
http://jcgt.org/published/0007/03/01/

Received:	2016-12-28		
Recommended:	2017-07-26	Corresponding Editor:	Natalya Tatarchuk
Published:	2018-07-26	Editor-in-Chief:	Marc Olano