Pacific Graphics (2022)
Y. Yang, A. D. Parakkat, B. Deng, and S.T. Noh (Editors)

Improving View Independent Rendering for Multiview Effects

Ajinkya Gavane and Benjamin Watson

North Carolina State University, USA

Figure 1: *Soft shadows (leftmost), cube map reflections for dining and gallery scenes [McG17] (middle two), and omnidirectional soft shadows (rightmost) rendered by improved view independent rendering (iVIR).*

Abstract

This paper describes improvements to view independent rendering (VIR) that make it much more useful for multiview effects. Improved VIR's (iVIR's) soft shadows are nearly identical in quality to VIR's and produced with comparable speed (several times faster than multipass rendering), even when using a simpler bufferless implementation that does not risk overflow. iVIR's omnidirectional shadow results are still better, often nearly twice as fast as VIR's, even when bufferless. Most impressively, iVIR enables complex environment mapping in real time, producing high-quality reflections up to an order of magnitude faster than VIR, and 2-4 times faster than multipass rendering.

CCS Concepts

• ***Computing methodologies*** → ***Rendering;*** *Graphics processors; Point-based models;*

1. Introduction

Computer graphics hardware has had difficulty with real-time rendering of multiview effects such as soft shadows, object reflections and depth of field, because they require multiple model traversals. Hardware ray tracing can now also produce these effects, but still struggles to sample them adequately [HAM19].

View-independent rasterization (VIR) renders multiview effects by using points as a display primitive, avoiding the complexity of multiple rendering passes [MWH17]. For every frame, it carefully transforms input triangles into a point cloud specialized to the current set of views. It then uses these points to render views in parallel, with an order of magnitude fewer passes over the geometry. Unfortunately, it requires use of a fixed-size point buffer, risking overflow. More importantly, it struggles in applications with disparate views that increase point cloud size (e.g., omnidirectional shadows and environment maps), and in applications with heavy shader loads (e.g., environment maps and depth of field).

This short paper presents our improvements to VIR, including:

- *Better sampling efficiency:* We introduce uniform sampling of triangles, per-triangle perspective correction and detailed methods for stochastic culling. For environment mapping, we also dynamically resize cube map buffers to eye resolution.
- *Better practicality:* When shader loads are small (e.g. soft and omnidirectional shadows), these efficiencies enable use of fragment rather than compute shaders fed by a point buffer, avoiding overflow concerns. In addition, with the new per-triangle perspective correction, projections can vary across views.
- *Broader application:* We use improved VIR (iVIR) not only for

soft shadows, but unlike VIR, also for omnidirectional soft shadows and dynamic cube-mapped reflections. When rendering:

- *soft shadows*, iVIR is more practical than VIR without significantly changing quality or speed. iVIR does not raise buffer overflow concerns and supports variable projections (e.g. for directional or spot lights). iVIR, VIR and multiview rendering (MVR) quality is nearly identical, while iVIR's speed remains similar to VIR's and several times faster than MVR's.
- *omnidirectional soft shadows*, iVIR has the same practical advantages as when rendering soft shadows, while also generating imagery 1.8× times faster than VIR and 1.5× times faster than MVR.
- *cube-mapped reflections*, iVIR generates imagery of similar quality to VIR and MVR, but does so up to 19× faster than VIR and up to 3.5× faster than MVR.

2. Related Work

Rendering realistic imagery requires accurate simulation of light flow. However, accurate sampling of the light flow integral [Kaj86] can be difficult [LAC*11]. With rasterization hardware, the fastest way to sample the integral is often MVR: storing many views in off-screen buffers and combining them. However, MVR remains slow, often requiring sparse sampling and filtering to reduce noise [SAC*11, HZP07, HZP06, HREB11, UKS*20, DSNS10]. But even the best of these techniques struggles when generating more than 32 views. Ray-tracing hardware has recently enabled hybrid rasterization-ray tracing pipelines for computing effects including shadows, reflections and refractions [SAGC*12, BBHW*19]. These techniques rely on stochastic sampling making them prone to noise, and require post-filtering resulting in blurry reflections.

2.1. Shadows and Shadow Mapping

Computer graphics renders "hard" shadows cast by point lights and the more realistic "soft shadows" are cast by area light sources. Shadow mapping [HH97] and shadow volume [AMA02] soft shadow solutions exist. The shadow mapping algorithm can be quite accurate, but is too slow for real-time use [ESAW16]. The shadow volume algorithm can be fairly fast, but is relatively complex and somewhat inaccurate. Many faster but more approximate techniques have also been introduced [HLH*03, AHL*06]. One example is percentage-closer soft shadows (PCSS), which adaptively filters hard shadow boundaries to produce penumbrae [RSC87, Fer05].

These algorithms work well for directional lights, which emit light only toward a small part of the environment around the light. However, in reality most light sources emit light in many directions. To generate omnidirectional shadows for a point light, we can create six separate shadow buffers, each defined by a face in a cube around the light, and use shadow mapping with each of those buffers [Ger04, Kas13]. To implement omnidirectional soft shadows, we must generate these six depth maps for each sample of the omnidirectional light source.

2.2. Reflections and Environment Mapping

Reflections can be achieved using ray tracing [Whi79], but ray-traced reflections can alias, particularly during animation. Avoiding aliasing either requires more samples and rays, or hybridized solutions using rasterized reflections [AMN19, BBHW*19].

The best-known rasterized solution is environment mapping, an MVR technique that turns images of the environment around the reflective object into texture maps that are accessed using view-dependent lookups. Several variations differ in the mapping between the rectangular textures and the reflective surface [BN76, Wil83, MIL84, HS98, ED08]. Perhaps the most widely used is cube mapping [Gre86], because it is a good fit to rectangular rasterization windows, permitting interactive reflections. Environment mapping produces convincing reflections in real time, but works best when the scene is static. In dynamic scenes, textures must be updated each frame. Moreover, when reflected objects are nearby, different reflective objects will "see" different views of that object, and cannot share one environment map. These shortcomings can require extensive multiview rendering in each frame.

2.3. Points and VIR rendering

To avoid rasterization's limitations for multiview rendering, VIR relies on points [LW85]. Triangles often outnumber pixels in today's applications, leading many to argue that points are a better rendering primitive [GP11]. Yet points are not used widely, since their discontinuity can create "holes" when views change. To avoid this, point renderers use dense point clouds that render slowly, or sparse clouds with complex reconstruction that again render slowly (e.g., [RGK*08]), or simply produce low-quality imagery.

To improve point rendering and support multiview rendering, VIR [MWH17] exploits rasterization hardware, which efficiently transforms triangles into points. For each frame, VIR generates a cloud of points customized to the current set of views in real time. It then renders the cloud in parallel into multiple views, reducing the number of geometry passes by a factor of ten. To accomplish this, for any triangle visible in at least one view, the geometry shader computes specialized viewing and projection matrices that center the triangle, orient it parallel to the view plane, and achieve a watertight sampling rate. It then applies the matrices to the triangle and rasterizes it to generate points. Next, the fragment shader writes each point to a buffer. When all points have been generated, the compute shader passes over this buffer, transforming and projecting each point into multiple views.

3. Improving View Independent Rasterization

We improve on Marrs et al.'s VIR [MWH17] to increase its practicality and efficiency, allowing broader application. These improvements include more efficient *uniform sampling* of triangles, *per-triangle perspective correction* that increases both sampling parsimony and rendering flexibility, and *detailed stochastic culling*, again improving sampling efficiency and speed. These efficiency improvements also often allow the use of a much more practical *bufferless* implementation, which relies only on fragment shaders rather than compute shaders fed by an intermediate point buffer

that might overflow. We highlight each of these improvements in this section with italics.

3.1. Watertight and Efficient Point Sampling

To produce the points that represent a triangle, iVIR rasterizes it in a view-independent fashion, with each fragment becoming a point. The primary function of the geometry shader is to set the triangle up for view-independent rasterization, producing points at a desired sampling rate.

iVIR begins by ensuring that the triangle is front-facing in at least one off-screen (e.g. shadow or reflection) view. If so, like VIR, iVIR centers the triangle in the viewport, and aligns it parallel to the view plane. To achieve this, the geometry shader computes and applies the T_{align} transformation matrix unique to each triangle, as shown in Eq. 1.

$$T_{align} = \begin{bmatrix} \hat{u}_x & \hat{u}_y & \hat{u}_z & -(\hat{u}\cdot\vec{c}) \\ \hat{v}_x & \hat{v}_y & \hat{v}_z & -(\hat{v}\cdot\vec{c}) \\ \hat{n}_x & \hat{n}_y & \hat{n}_z & -(\hat{n}\cdot\vec{c}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where $\hat{u}$, $\hat{v}$ and $\hat{n}$ are the mutually orthogonal vectors forming the basis frame of a triangle; $\hat{n}$ is the triangle unit normal vector; and the vector $\vec{c}$ contains the coordinates of the triangle centroid. Any point on the triangle can be used to locate the frame, as long as the triangle lies completely inside the viewing volume after transformation with T_{align}. Applying the T_{align} matrix places the camera at the origin, looking down the negative z-axis toward the triangle.

Marrs et al.'s VIR computed the watertight multiview sampling rate s_{mv} by assuming that the eye, off-screen, and point-sampling views were all perspective, using the same fields of view. Unfortunately, point sampling in perspective leads to sampling inefficiencies, with points spread unevenly across each triangle due to perspective distortion. Moreover, these assumptions reduce rendering flexibility. We address these problems by using orthogonal projection in the point-sampling view, resulting in *uniform sampling* across the triangle.

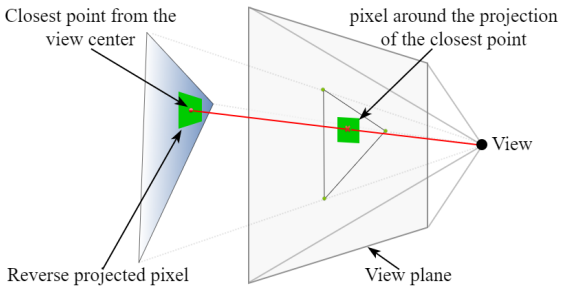


Figure 2: *Reverse projection of a pixel area around the closest point on the polygon from the eye.*

Use of different (orthogonal and perspective) projections in the eye, off-screen and point-sampling views means that distance alone cannot indicate the required sampling rate. iVIR finds ρ_{mv}, the maximum point density needed on the triangle's surface, as illustrated in Figure 2. For each triangle t, we first find the closest point on the triangle from a given view v [Ebe99]. This point has the highest sample density on the triangle for that view. We compute the area of a reverse-projected pixel centered on that point $area_{P,v,t}$ by reverse-projecting its corners onto the polygon in model space. Across all views, the maximum multiview sampling density ρ_{mv} is given by the Eq. 2, and the orthogonal scaling factor S_{ortho} is given by Eq. 3.

$$\rho_{mv} = \forall_{v\in V}\, max(\rho_{mv}, \frac{area_t}{area_{P,v,t}}) \tag{2}$$

$$S_{ortho} = \sqrt{\frac{\rho_{ortho}}{\rho_{mv}}} \tag{3}$$

where V is the set of all view centers of destination views, $area_t$ is the area of the triangle in model space, and ρ_{ortho} is the sampling density for VIR's orthographic projection, which depends on the chosen viewing volume. Note that while VIR used a global worst-case calculation of perspective distortion to conservatively increase sampling rate and ensure watertight sampling, iVIR's method described here implicitly includes a much more efficient *per-triangle perspective correction*, which also improves rendering flexibility.

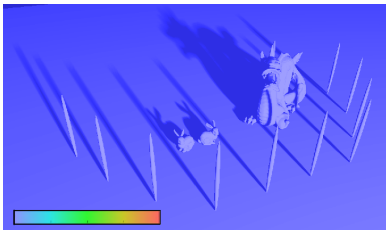


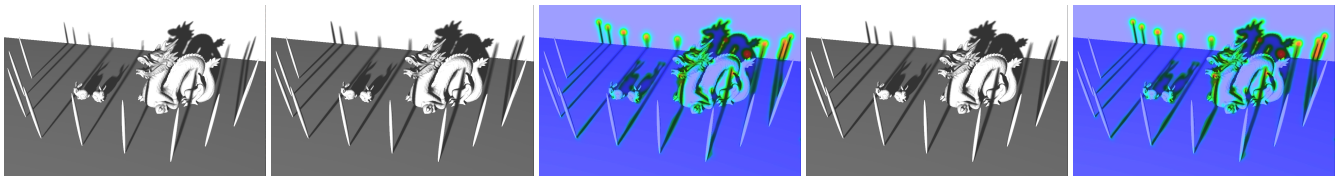
Figure 3: *A perceptual comparison of iVIR with and without stochastic culling. Red indicates a more perceivable difference. HDR-VDP2 shows little or no difference.*

To ensure the current triangle will be sampled at this rate, we simply scale it by S_{ortho}. We define iVIR coordinate space as world space coordinates, transformed by T_{align}, and scaled by orthogonal scaling factor S_{ortho}, as shown in the Eq. 4. We apply this composite transform to each triangle vertex. Because of the reverse projection of pixel area on the triangle around the point with highest sample density, our orthogonal sampling implicitly includes per-triangle perspective correction.

$$T_{ivir} = scale(S_{ortho}) * T_{align} \tag{4}$$

3.2. Stochastic Culling

To improve speed further, we stochastically cull (and avoid generating points for) triangles that span less than $1/10^{th}$ of a pixel in VIR's point generation view. The smaller the area of the triangle A_t, the more likely it will be culled, with probability $P_c = 1 - 10A_t$. Stochastic culling breaks our watertight guarantee, but we have not yet observed any "holes" resulting in practice, and we estimate their probability to be extremely unlikely: less than one chance in a million for shadows; even lower for reflections. For example, Figure 3 shows a perceptual comparison of improved VIR with and without culling using HDR-VDP2 [MKRH11]; cool heatmap colors indicate little or no difference. Across this same range of models, we



(a) *MVR* **(b)** *Marrs et al.'s VIR* **(c)** *MVR vs Marrs et al.'s VIR* **(d)** *iVIR* **(e)** *MVR vs iVIR*

Figure 4: *Soft shadows generating* 128 *depth maps, rendered using (a) MVR, (b,c) Marrs et al.'s VIR, and (d,e) iVIR. MVR takes* 57*ms, VIR takes* 10.4*ms and iVIR takes* 16.6*ms using bufferless and* 9.8*ms with buffered implementation. (c,e) HDR-VDP2 imagery are compared against MVR, with red indicating that differences are very perceivable, and blue rarely perceived.*

found that stochastic culling is most beneficial when subpixel triangles are common. This replicates Marrs et al's [MWH17] finding that for large triangles (spanning dozens of pixels or more), VIR is less efficient than standard rasterization.

4. Soft Shadow using iVIR

We demonstrate the practicality and potential of improved VIR with soft shadows. Below, we offer comparisons to both high quality and high-speed shadow algorithms, as well as a comparison to Marrs et al.'s implementation [MWH17].

GPU Performance of iVIR [with stochastic culling] Soft Shadows for 128 Views

Models (# tris)	Improved VIR				Marrs et al.'s VIR			MVR time
	#points	pt gen time	Bufls time	Bufd time	#points	pt gen time	total time	
Tree House (151.8K)	245.5K [194.1K]	0.69	3.33 [1.82]	3.81 [3.50]	253K [203.7K]	0.67	4.01 [3.77]	7.33
Dragon (883.3K)	693.7K [482.1K]	4.67	16.73 [10.86]	9.79 [8.83]	935.1K [810.4K]	4.08	10.39 [9.73]	57.01
Buddha (1.1M)	569.5K (130.5K)	5.35	20.64 [9.01]	9.70 [7.75]	617.2k [252.1K]	4.65	9.24 [7.58]	69.33
Lucy (2.0M)	1.0M (125.9K)	9.85	38.35 [14.05]	15.92 [12.21]	1.1M [298.6K]	8.28	14.75 [11.38]	122.3

Table 1: *Soft shadow performance of iVIR, VIR, and MVR for different models ranging from 150k to 2M triangles generating 128 depth buffers of* 1024^2 *resolution. All GPU times are measured in milliseconds (ms). Fastest times highlighted in blue and the results using stochastic culling are in brackets ([]).*

4.1. High Quality Comparision

As an evaluation platform, we used OpenGL 4.5 on a PC with an Intel i7-8700K @ 3.70 GHz CPU and an NVIDIA 1080Ti GPU, running Windows 10 OS. We rendered multiple dynamic scenes, rotating around themselves, while lights remained stationary, casting moving shadows. We used 32-bit unsigned depth buffers, with a resolution of 1024^2. For each light source sample, we set field of view to 45^o.

Like Marrs et al., we produced 128 views in four passes (32 per pass, the warp size of our GPU). As a high quality comparison, we used multiview rendering MVR, which used 128 passes to create 128 standard shadow maps [Wil78]. To compare the performance of these methods, we averaged GPU run-time and the number of points generated over 1000 frames of execution.

Table 1 shows results for several models [McG17]. The leftmost column shows the number of triangles per model. The adjacent four show improved iVIR's point cloud size, the time required to generate that point cloud, and total time to generate point cloud and construct depth maps (with results including stochastic culling in brackets) for bufferless and buffered implementation. For comparison, the next three columns reports the Marrs et al.'s point cloud size, the time required to generate that point cloud, and the total time to generate point cloud and construct depth maps (with results including stochastic culling in the brackets). The last column reports the total time taken by MVR to make depth maps. The illumination technique is the same for iVIR, VIR, and MVR, we do not include it. iVIR renders these dynamic, complex soft shadows up to 3.4 times faster than MVR without stochastic culling, and up to 7 times faster with it.

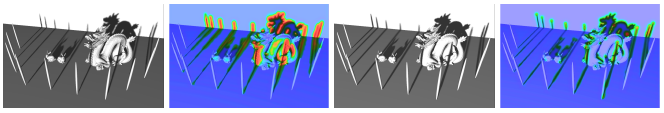
Our improvements had minimal impact on iVIR's performance and image quality producing nearly identical soft shadows as VIR at similar speed. While our orthogonal projection sampling generated fewer points than Marrs et al.'s sampling rate, it required more time to do so, particularly in models with more triangles. For example, when generating 128 views for a model with 2 million triangles, orthogonal projection sampling generated 126K points in 9.85ms, whereas Marrs et al.'s sampling generated 299kK points in 8.28 ms.

Figure 4 shows soft shadows generated by iVIR, VIR and MVR. Though iVIR is faster than MVR, its visual quality is quite similar to high quality MVR and VIR, and stable under animation. iVIR includes the hallmarks of high quality shadows, such as soft penumbras and contact hardening (sharper shadows closer to the light). Because iVIR and MVR both use shadow mapping and differ only in how they generate depth buffers, both suffer the same artifacts (e.g. "peter panning" and acne). Note that the breaks in the dragon's shadow with iVIR are smaller than in MVR; iVIR's view independent samples silhouettes more densely than view dependent MVR.

4.2. High Speed Comparison

To evaluate the use of iVIR in a more practical, real time setting, we compare iVIR to percentage-closer soft shadows (PCSS) [Fer05]. We generated soft shadows using 16 views in 2.6 ms and compared it to an image generated by PCSS with 96 samples per pixel (32 blockers and 64 filter samples), in 2.5 ms. The resulting images are shown in Figure 5 along with their perceptual comparison against a

reference 128-view MVR solution using HDR-VDP2 [MKRH11]. The image generated by iVIR has less error than PCSS, especially at the region where the rods cast shadows on the dragon.



(a) *PCSS* **(b)** *MVR vs PCSS* **(c)** *iVIR* **(d)** *MVR vs iVIR*

Figure 5: *Soft shadow rendered by (a) PCSS (32 blocker and 64 PCF samples) and (c) iVIR (16 views). (b) and (d) are the HDR-VDP2 perceptual comparison of PCSS and iVIR against 128-view MVR respectively. PCSS takes 2.5 ms, whereas iVIR takes 2.6 ms delivering better quality.*

5. Environment Mapping using iVIR

In this section, we describe our application of iVIR to environment mapping. We first emphasis on iVIR's ability to adapt to different off-screen view resolution, and continue examining the speed and image quality of our iVIR-based implementation to VIR and MVR

5.1. Adapting iVIR to Environment Mapping

iVIR samples a triangle by reverse-projecting a pixel area on the view plane on to the triangle, making it adaptive to the off-screen view resolution. Thus, reducing the required point density and ultimately, point cloud size. We adjust the sizes of these off-screen views in every frame by projecting the corners of reflective object's bounding box on the eye's view.

We experimented with both bufferless implementation in which the fragment shader writes points directly to the off-screen views, and compute implementation where we store point in a point buffer for processing in compute. While both of them produced identical imagery, bufferless implementation was consistently less than half as fast. Schutz et al. [SKW21] reported similar results.

Scene	#Refl Objs (#tris)	Improved VIR			Marrs et al.'s VIR		MVR time
		iVIR (#points)	Bufferless tot time	Buffered tot time	VIR (#points)	Total Time	
Dining Scene	1 (1.57M)	1.46 (1.28M)	2.68	2.60	13.34 (15.07M)	17.01	3.94
	8 (1.92M)	4.30 (3.82M)	16.69	11.63	98.91 (40.42M)	199.95	21.31
	16 (2.34M)	8.43 (5.57M)	43.84	24.17	204.51 (63.37M)	495.35	48.06
	20 (2.39M)	10.30 (5.60M)	56.53	29.65	206.61 (66.88M)	570.10	60.30
Gallery Scene	1 (1.01M)	1.13 (352.53K)	2.20	2.00	8.73 (10.26M)	11.40	3.44
	8 (1.09M)	3.37 (859.33K)	17.12	8.94	34.33 (17.63M)	68.25	28.64
	16 (1.19M)	6.27 (995.82K)	38.88	17.35	61.46 (23.72M)	191.46	58.20
	20 (1.24M)	7.90 (1.06M)	50.90	21.46	63.86 (25.49M)	227.04	74.49

Table 2: *Environment mapping performance table comparing iVIR, VIR and MVR for the dining and gallery scenes with 1-20 reflective objects and 1.5M and 1.0M non-reflective triangles, respectively. All GPU times are measured in milliseconds (ms). Fastest times are highlighted in blue, and slowest in red.*

Unlike soft shadows, environment mapping requires multiple storage buffers to store point data. The buffer must contain point center, normal, material data, and it's optionally visibility information in each of cubemap's faces. We currently use 3 buffers: point center in a vector3 buffer, normal and any materials are in a uvec4 buffer,and the visibility string is in two 64-bit unsigned integer buffers.

The compute shader processes all points in the storage buffers, and writes the point into that view's off-screen buffer. Our cube maps used 32-bit unsigned integer buffers, with a maximum adaptive resolution of 512^2. The compute shader cannot use Z-buffering hardware, with its guarantee of atomic data access by parallel threads. Instead, we use atomic operations on integer buffers. Each off-screen buffer pixel is a 32-bit unsigned integer with the 13 most significant bits used for storing the depth of the point, and the remaining 19 bits for storing the color (6-7-6 bit of red, green, and blue). High quality environment maps require filtering to accurately represent reflections. Unfortunately, hardware filtering support such as mipmapping is not available to compute shaders. Thus, for each off-screen view, we carry an image pass to copy it in a mipmapped buffer.

5.2. Results

We used the same testing configuration as discussed in 4.1. For testing, we used a dining scene and gallery scene [McG17], shown in Figure 6. Scenes were dynamic, with the eye revolving around the scene, and all the reflective objects moving in a periodic fash-

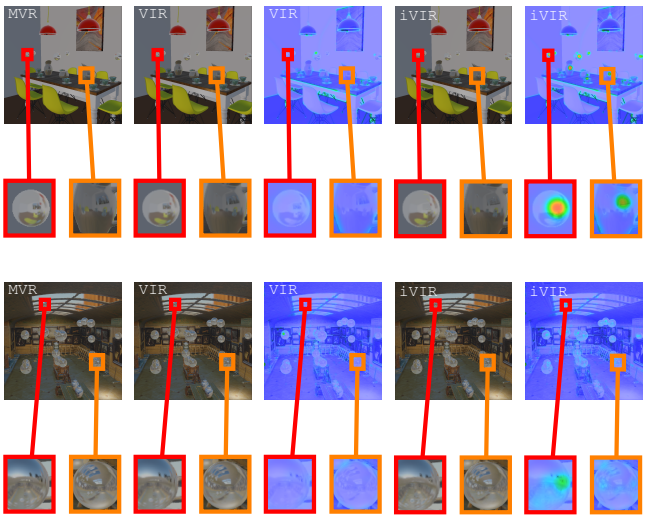


Figure 6: *Environment cubemapped reflections rendered by MVR (leftmost column), VIR (next two columns, with HDR-VDP2 perceptual comparison), and iVIR (last two columns). All HDR-VDP2 imagery compare against MVR, with red indicating that differences are very perceivable, and blue rarely perceived.*

#light samples	iVIR			Marrs et al.'s VIR		MVR time
	#points	Bufferless tot time	Buffered tot time	#points	total time	
1	10.94M	2.46	4.03	14.31 M	4.75	2.16
5	11.43M	4.86	5.86	14.83 M	7.11	7.22
10	11.69M	7.88	7.98	15.03 M	8.98	15.15
20	11.74M	20.32	30.74	15.07 M	36.97	29.74

Table 3: *Omnidirectional soft shadow performance results comparing iVIR, VIR, and MVR for the dining scene with* 1.4*M triangles. As the number of light samples varies, we report total time (in ms) and point cloud size. Fastest times are in blue, slowest in red.*

ion. We used a physically-based rendering shader using roughness and metallic textures [PJH16]. In our buffered implementation, our cube maps used 32-bit unsigned integer buffers, with a maximum adaptive resolution of 512^2.

Table 2 shows results for the dining and gallery scenes [McG17]. The leftmost column is the name of the scene, and the adjacent column to it shows the number of reflective objects in the scene and the corresponding triangles per scene. The adjacent three show improved VIR's point generation time (with number of points in parentheses), total rendering time to generate cubemaps for bufferless, and compute iVIR. For comparison, the next two column reports Marrs et al.'s point generation time (with number of points in parentheses), total rendering time to generate cubemaps. The last column reports MVR total time to generate cubemaps. The deferred shading pass is the same for iVIR, VIR, and MVR, we do not include it. The bufferless version of iVIR is 1.5× faster than MVR, while the compute version of iVIR is up to 3.5× faster thanMVR.

Figure 6 shows environment mapped reflection generated by improved VIR and MVR. The first column shows reflections by MVR, the next two column shows reflection by VIR and the HDR-VDP2 [MKRH11] visual difference between VIR and MVR, and the last column shows reflection by iVIR and its visual difference between MVR. The first two rows shown the dining scene, while the last two show the gallery. First, in the red boxes of the second row, note the window's reflection on the sphere: iVIR shows window blinds, while MVR and VIR shows the view outside the window. Because iVIR renders at adaptive resolution and with conservative rasterization a point smaller than the buffer pixel size occupies the entirety of the pixel, while in VIR and MVR the buffer resolution is set to 512^2 and mipmapped. The RMSE measure between iVIR and MVR are 0.358 and 0.490 for dining and gallery scene respectively, and for VIR and MVR are 0.307 and 0.374 for dining and gallery scene respectively (note that the minor differences outside of reflections are due to different anti-aliasing techniques used for iVIR and VIR (FXAA), and MVR (MSAA)). iVIR is similar in quality to VIR while generating 95% less points and is 10× faster using bufferless, and 19× faster using compute.

6. Omnidirectional Soft Shadows using iVIR

We further gauge our improvements by applying iVIR to omnidirectional soft shadows, and comparing results with VIR and MVR. We used the same testing configuration described in 4.1. Figure 7 shows that these three algorithms produce imagery of similar quality. Table 3 shows rendering times for the dining scene with 1.4*M* triangles. The leftmost column shows the number of light samples on a spherical omnidirectional light. The next column shows iVIR's point cloud size. The third and fourth show the total time for iVIR to generate point clouds and make shadow cube maps using bufferless and buffered implementations, respectively. The next two columns report VIR's point cloud size and total rendering time, while the last column reports MVR's total time. iVIR generated 22% fewer points than VIR, and its bufferless version is 1.8× faster than VIR and 1.46× faster than MVR.

(a) *MVR* **(b)** *Marrs et al.'s VIR* **(c)** *iVIR*

Figure 7: *Omnidirectional soft shadows rendered using (a) MVR, (b) VIR, and (c) iVIR. The omnidirectional light is spherical, with 20 samples on its surface. In this figure only, the light is shown in cyan and twice its actual size for illustrative purposes.)*

7. Conclusions, Limitations, and Future Work

This paper describes iVIR, an improved version of to Marrs et al.'s VIR. Marrs et al. demonstrated only an application of VIR to soft shadows. iVIR allows not only a more practical soft shadow implementation of similar speed and quality, but unlike VIR also enables real-time omnidirectional soft shadows and complex reflections with cube mapping. iVIR's environment mapping implementation is particularly impressive, rendering high-quality reflections 3.5× faster than MVR, and 19× faster than VIR.

Yet iVIR does have limitations. Its stochastic triangle culling breaks the guarantee of watertight sampling, though we could not see holes in our testing. Also, iVIR can exaggerate fine (single pixel or smaller) detail, because it uses conservative rasterization and when environment mapping, coarse cube map textures adapted to the resolution of the eye's view. In the near term, we will mitigate this problem by culling points that lie outside of triangles, and by being less aggressive in coarsening cube map textures. Longer term, we plan to apply iVIR to defocus blur and light field displays, and to study global probabilistic limits for potential holes resulting from stochastic culling.

8. Acknowledgments

This work was supported by the National Science Foundation under award IIS-2008590.

References

[AHL*06] ATTY L., HOLZSCHUCH N., LAPIERRE M., HASENFRATZ J.-M., HANSEN C., SILLION F. X.: Soft shadow maps: Efficient sam-

pling of light source visibility. In *Computer graphics forum* (2006), vol. 25(4), Wiley Online Library, pp. 725–741. 2

[AMA02] AKENINE-MÖLLER T., ASSARSSON U.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. *Rendering Techniques 2002* (2002), 297–306. 2

[AMN19] AKENINE-MÖLLER T., NILSSON J.: Simple environment map filtering using ray cones and ray differentials. In *Ray Tracing Gems*. Springer, 2019, pp. 347–351. 2

[BBHW*19] BARRÉ-BRISEBOIS C., HALÉN H., WIHLIDAL G., LAURITZEN A., BEKKERS J., STACHOWIAK T., ANDERSSON J.: Hybrid rendering for real-time ray tracing. In *Ray Tracing Gems*. Springer, 2019, pp. 437–473. 2

[BN76] BLINN J. F., NEWELL M. E.: Texture and reflection in computer generated images. *Commun. ACM 19*, 10 (1976), 542–547. 2

[DSNS10] DE SORBIER F., NOZICK V., SAITO H.: Gpu-based multiview rendering. In *Computer Games, Multimedia and Allied Technology* (2010), pp. 7–13. 2

[Ebe99] EBERLY D.: Distance between point and triangle in 3d. *Magic Software* (1999). URL: http://www.magic-software.com/Documentation/pt3tri3.pdf. 3

[ED08] ENGELHARDT T., DACHSBACHER C.: Octahedron environment maps. In *VMV* (2008), Citeseer, pp. 383–388. 2

[ESAW16] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-time shadows*. AK Peters/CRC Press, 2016. 2

[Fer05] FERNANDO R.: Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches* (2005), ACM, p. 35. 2, 4

[Ger04] GERASIMOV P.: Omnidirectional shadow mapping. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics 20* (2004), 193–204. 2

[GP11] GROSS M., PFISTER H.: *Point-based graphics*. Elsevier, 2011. 2

[Gre86] GREENE N.: Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications 6*, 11 (1986), 21–29. 2

[HAM19] HAINES E., AKENINE-MÖLLER T. (Eds.): *Ray Tracing Gems*. Apress, 2019. http://raytracinggems.com. 1

[HH97] HECKBERT P. S., HERF M.: *Simulating soft shadows with graphics hardware*. Tech. rep., Carnegie Mellon Univ Pittsburgh PA Dept of Computer Science, 1997. 2

[HLH*03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F., GRAVIR A.: A survey of real-time soft shadows algorithms. In *Computer Graphics Forum* (2003), vol. 22(4), Wiley Online Library, pp. 753–774. 2

[HREB11] HOLLANDER M., RITSCHEL T., EISEMANN E., BOUBEKEUR T.: Manylods: Parallel many-view level-of-detail selection for real-time global illumination. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 1233–1240. 2

[HS98] HEIDRICH W., SEIDEL H.-P.: View-independent environment maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (1998), pp. 39–ff. 2

[HZP06] HÜBNER T., ZHANG Y., PAJAROLA R.: Multi-view point splatting. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (2006), pp. 285–294. 2

[HZP07] HÜBNER T., ZHANG Y., PAJAROLA R.: Single-pass multiview rendering. *IADIS International Journal on Computer Science and Information Systems 2*, 2 (2007), 122–140. 2

[Kaj86] KAJIYA J. T.: The rendering equation. In *ACM Siggraph Computer Graphics* (1986), vol. 20(4), ACM, pp. 143–150. 2

[Kas13] KASYAN N.: Playing with real-time shadows. *SIGGRAPH: ACM SIGGRAPH 2013 Courses* (2013). 2

[LAC*11] LEHTINEN J., AILA T., CHEN J., LAINE S., DURAND F.: Temporal light field reconstruction for rendering distribution effects. In *ACM Trans. Graphics* (2011), vol. 30(4), ACM, p. 55. 2

[LW85] LEVOY M., WHITTED T.: *The use of points as display primitives (Technical Report TR 85-022)*. 1985. 2

[McG17] MCGUIRE M.: Computer graphics archive, July 2017. URL: https://casual-effects.com/data. 1, 4, 5, 6

[MIL84] MILLER G.: Illumination and reflection maps: Simulated objects in simulated and real environments. *ACM SIGGRAPH'84 course notes* (1984). 2

[MKRH11] MANTIUK R., KIM K. J., REMPEL A. G., HEIDRICH W.: Hdr-vdp-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions. *ACM Trans. Graphics 30*, 4 (2011), 40. 3, 5, 6

[MWH17] MARRS A., WATSON B., HEALEY C. G.: Real-time view independent rasterization for multi-view rendering. In *Eurographics (Short Papers)* (2017), pp. 17–20. 1, 2, 4

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. 6

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM transactions on graphics (tog) 27*, 5 (2008), 1–8. 2

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *ACM Siggraph Computer Graphics* (1987), vol. 21(4), ACM, pp. 283–291. 2

[SAC*11] SHIRLEY P., AILA T., COHEN J., ENDERTON E., LAINE S., LUEBKE D., MCGUIRE M.: A local image reconstruction algorithm for stochastic rendering. In *Symp. Ictv. 3D Graphics & Games* (2011), ACM, p. 5. 2

[SAGC*12] SABINO T. L., ANDRADE P., GONZALES CLUA E. W., MONTENEGRO A., PAGLIOSA P.: A hybrid gpu rasterized and ray traced rendering pipeline for real time rendering of per pixel effects. In *International Conference on Entertainment Computing* (2012), Springer, pp. 292–305. 2

[SKW21] SCHÜTZ M., KERBL B., WIMMER M.: Rendering point clouds with compute shaders and vertex order optimization. *arXiv preprint arXiv:2104.07526* (2021). 5

[UKS*20] UNTERGUGGENBERGER J., KERBL B., STEINBERGER M., SCHMALSTIEG D., WIMMER M.: Fast multi-view rendering for real-time applications. In *EGPGV@ Eurographics/EuroVis* (2020), pp. 13–23. 2

[Whi79] WHITTED T.: An improved illumination model for shaded display. In *ACM SIGGRAPH Computer Graphics* (1979), vol. 13(2), ACM, p. 14. 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics* (1978), vol. 12(3), ACM, pp. 270–274. 4

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (1983), pp. 1–11. 2