Real-Time View Independent Rasterization for Multi-View Rendering

Adam Marrs[†], Benjamin Watson, and Christopher G. Healey

North Carolina State University

Abstract

Existing graphics hardware parallelizes view generation poorly, placing many multi-view effects – such as soft shadows, defocus blur, and reflections – out of reach for real-time applications. We present emerging solutions that address this problem using a high density point set tailored per frame to the current multi-view configuration, coupled with relatively simple reconstruction kernels. Points are a more flexible rendering primitive, which we leverage to render many high resolution views in parallel. Preliminary results show our approach accelerates point generation and the rendering of multi-view soft shadows up to 9×.

1. Introduction

Highly dynamic real-time applications, such as interactive games, now demand image quality exceeding that which direct illumination approaches and precomputation strategies are capable of delivering. Accurate light simulation accounts for photons arriving not only from direct sources, but also indirect sources, which requires the evaluation of complex multi-dimensional integrals [Kaj86]. Today, due to decades of investment in graphics hardware designed primarily to simulate direct illumination, sampling techniques that *serially rasterize multiple views* are still commonly employed to approximate these integrals [Gre86, HA90, Hak01]. Although existing fixed function rasterization hardware is highly optimized, the bias towards creating a single view in a single graphics pipeline execution severely limits performance in multi-view rendering scenarios. Existing graphics hardware *parallelizes view generation poorly*, making multi-view effects such as soft shadows, defocus blur, and reflections especially challenging to render efficiently.

The use of points as the primary rendering primitive has been demonstrated in offline systems as a viable strategy to accelerate the rendering of complex multi-view effects including ambient occlusion and diffuse global illumination [GP07]. To avoid visible "holes" in rendered geometry, these techniques use dense sampling and/or complex splatting and reconstruction methods. Point-based rendering is rarely used for real-time multi-view effects since 1) *dense point sets* appropriate for all views cannot be regenerated in real-time for animation, nor be rendered to many views within real time frame budgets; while 2) *sparse point sets* require elaborate splatting and reconstruction algorithms that do not parallelize effectively on existing GPUs, and ultimately suffer from low quality approximations of the original geometry.

In this short paper, we present emerging solutions that tackle these problems. Modern graphics hardware solves direct illumination by coupling dense sampling tailored to the current view with extremely simple reconstruction that parallelizes well, fully exploiting the GPU's raw power. Inspired by this approach, we propose solving indirect illumination using a *high density point set tailored per frame* to the current multi-view configuration, coupled with relatively *simple reconstruction* kernels. Preliminary results show that our approach can transform two million polygons into a high density point cloud – specialized for up to 128 views – in real time. Additionally, by restructuring the rendering computation using these points, we are able to render many high resolution views in parallel. In the modern GPU, points are a much more flexible primitive than triangles, allowing projection into many buffers in one pass. We demonstrate this by accelerating the rendering of depth maps used to compute multi-view soft shadows. We conclude with a discussion of the significant remaining challenges.

2. Related Work

Point-Based Rendering is based on the insight that the efficiency gained from a polygonal surface representation is no longer advantageous when a polygon covers only a single pixel (or less) of the output image [LW85]. In recent years, the number of pixels covered by the average polygon has decreased dramatically and the connectivity information of polygonal representations is less important to real-time efficiency than ever before [GP07].

Acquiring points has traditionally been performed offline by laser scanners [RL00]. For many real-time applications, points

[†] email: acmarrs@ncsu.edu

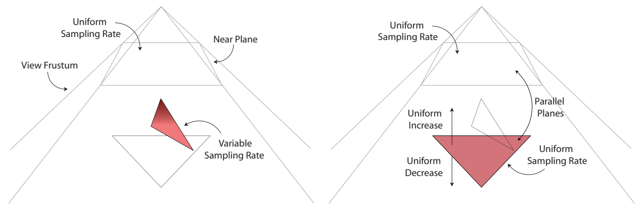

Figure 1: *Left: the view plane is uniformly sampled and the plane of a polygon in the view frustum receives variable sampling under perspective projection. Right: a special case, where the view plane and all parts of the polygon have uniform sampling rates when parallel, differing only by a ratio proportional to distance.*

must be generated at runtime from animated polygonal models. Early work produced points in real time by rendering many orthogonal views of each object [GD98]. Later, Bærentzen used depth peeling in three axis-aligned directions to capture multiple depth layers in each view using the GPU [Bær05]. More recently, an extension of Imperfect Shadow Mapping (ISM) used the GPU's tessellation unit [BBH13]. While faster than depth peeling, the tessellation pipeline is less efficient than the fixed function rasterization hardware utilized by our solution.

Managing sampling density (level of detail) is a challenging problem for all point-based approaches. Blending between polygons and points [DVS03] and constructing hierarchical acceleration structures [HREB11] have all been proposed. Complicated splatting, hole-filling, and interpolation techniques are used when points are sparse [MKC07,GP07]. Most of these solutions were designed for GPUs from a prior decade and are ill-suited to modern game engines.

3. View Independent Rasterization

The fixed function rasterization hardware of modern GPUs are highly-optimized point generation machines. By design, these hardware units produce *view-dependent* samples of input geometry by transforming all polygons of a *single view* using a *single view-projection plane*. The result is uniform sampling across the view plane and variable sampling across each projected polygon. See the left side of Figure 1 for an illustration.

A special case exists when the view plane and the plane of a transformed polygon are *parallel*. Shown on the right of Figure 1, the sampling rate is *uniform* across both planes, and only differs by a ratio proportional to the distance between the planes. In this situation, the rasterizer can be utilized as a powerful *view-independent point generator* by ensuring the convex hull of a parallel polygon fits within the view volume before rasterization. Increasing the projected area *uniformly increases* sampling density, while decreasing projected area *uniformly decreases* sampling.

We leverage this insight to generate a high density point cloud useful for *many views* in real time. We refer to this novel point generation approach as *View Independent Rasterization (VIR)*. By computing a view-projection plane *unique to each polygon* in the geometry stage of the graphics pipeline, VIR achieves its real-time performance by traversing the source geometry only once and sampling with fast, fixed-function rasterization hardware.

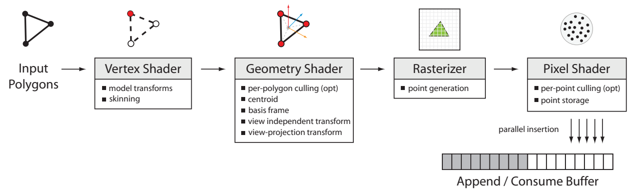

Figure 2: *View Independent Rasterization as implemented in the graphics pipeline.*

3.1. View Independent Transformation

To perform *view independent rasterization* of arbitrary polygons, we compute and apply a unique transformation matrix, T_{vir}, to each polygon in the Geometry Shader. Alternatively, T_{vir} can be computed and applied in the Hull Shader; however this requires tessellation support, prevents surfaces from being tessellated by the hardware, and incurs unnecessary overhead by activating the tessellation pipeline for all polygons. The transform centers each polygon about the z-axis, aligns the polygon's plane parallel to the X-Y plane, and is written as the 4×4 matrix:

$$T_{\text{vir}} = \begin{bmatrix} \widehat{u}_x & \widehat{u}_y & \widehat{u}_z & -(\widehat{u}\cdot c) \\ \widehat{v}_x & \widehat{v}_y & \widehat{v}_z & -(\widehat{v}\cdot c) \\ \widehat{n}_x & \widehat{n}_y & \widehat{n}_z & -(\widehat{n}\cdot c)+d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where $\widehat{n}$, $\widehat{u}$, and $\widehat{v}$ are mutually orthogonal unit vectors forming the polygon's *basis frame*. $\widehat{n}$ is the polygon's unit length geometric normal and the vector c is the translation of the polygon's centroid to the world origin. d is the desired distance from the polygon to the near plane, which uniformly affects sampling density. We discuss this further at the end of this section.

Next, we apply a default view-projection transformation to each polygon that positions the camera at the world origin looking down the positive z-axis. We store points generated by the rasterizer using the Pixel Shader stage. In this step, culling of point samples occurs to further tailor the point set to the current frame's multiview requirements. This can be accomplished using view-frustum and/or back-face comparisons before point storage. For simplicity, we store points in an unstructured, linear (D3D11 Append/-Consume) buffer; however, an arbitrary data structure may be used. These steps are illustrated in Figure 2.

Our preliminary implementation uses a conservative sampling density that varies per-polygon. Shown in Figure 3, the sampling density of a polygon processed by VIR is represented by a single distance value. To ensure a polygon is sampled sufficiently – placing at least one sample in each pixel in all relevant views – we compute the shortest distance, d, from the polygon's surface to the set of relevant view centers. This problem can be efficiently solved with a two dimensional projection [Jon95].

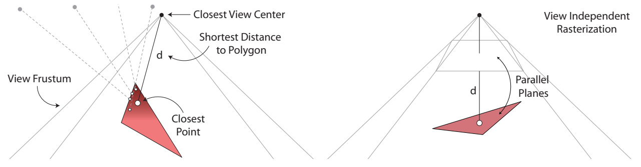

Figure 3: *Conservative multi-view sampling density is computed using the shortest distance from the polygon to all relevant views.*

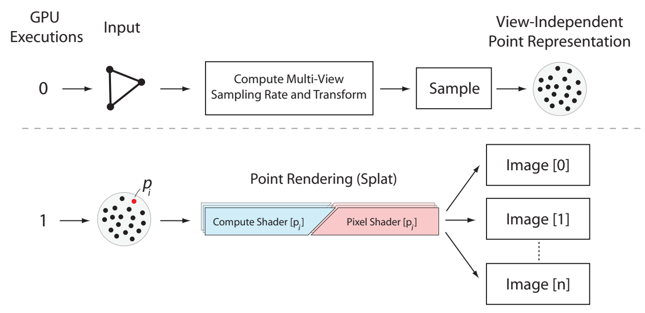

Figure 4: *Multi-view rendering using VIR restructures computation to improve point reuse and parallelization on existing GPUs.*

4. Parallel View Rendering

After point generation, we render multiple views in parallel using the point representation. We perform point rendering by either a) streaming points directly to the Pixel Shader stage of VIR, or b) storing points to a separate buffer and dispatching GPU compute threads (shown in Figure 4). Our simple point rendering kernels read a point's world-space location, and then for each view, apply the corresponding view-projection matrix, snap the projected location to the nearest neighbor pixel in the view's buffer, and perform z-buffering. Atomic functions resolve race conditions caused by multiple points projecting to the same texel.

5. Preliminary Results

Table 1 shows initial point generation performance results for polygonal models of varying complexity. Our experimental environment uses Windows 8.1 and Direct3D 11.1 running on an Intel i7-4790k @ 4.0 GHz with a NVIDIA Maxwell Titan X GPU. GPU times are averaged from 1,000 frames of execution.

VIR processes two million polygons, generating nearly 600,000 points tailored to 128 depth map views in only 5.48 milliseconds. Since each depth map view is 1024^2 resolution (averaging two input polygons per pixel), VIR dynamically adjusts the total number of points lower than the number of input polygons. The rendering of depth maps (including point generation) is accelerated up to 9× compared to traditional multi-pass rasterization. Points parallelize multi-view rendering more effectively than polygons, since they avoid the slow instancing or amplification operations required to parallelize polygon rendering across dozens of views. Further, we observe notable performance improvements using the GPU compute approach, which warrants further analysis. Shown in Figure 5, the visual quality of the soft shadows produced by our approach is excellent. A minor "bloating" of shadows occurs compared to rasterization, caused by our nearest neighbor reconstruction writing to texels that traditional polygonal rasterization ignores when the central sample of a texel is not covered (see magnified inlays).

VIR Point Generation and Depth Map Rendering (compared to Multi-Pass Rasterization) for 128 Views

Model	Triangles	VIR	Points	Depth Map Rendering	× Faster
Lucy	53,715	0.46	83,989	2.10 (2.21)	1.05
	106,973	0.62	125,136	2.69 (4.10)	1.52
	256,621	1.03	229,840	4.57 (15.91)	3.48
	502,667	1.95	369,328	7.51 (32.73)	4.36
	1,023,609	3.31	563,878	11.63 (65.86)	5.66
	2,005,071	5.48	599,916	13.03 (120.1)	9.22
Buddah	1,087,476	2.93	164,831	5.43 (52.80)	9.72
Trees	1,597,951	3.77	121,768	5.67 (37.00)	6.53

Table 1: *VIR point generation times (orange) for up to 2M polygons and 128 views. Depth Map rendering using points (including VIR point generation time) is compared to multi-pass rasterization of polygons (in parentheses). All times measured in milliseconds.*

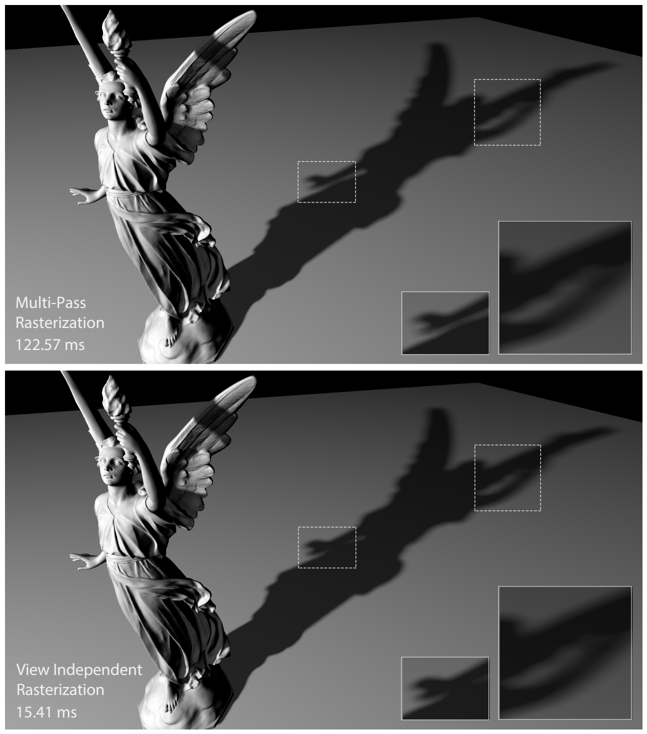
Multi-Pass Rasterization 122.57 ms
View Independent Rasterization 15.41 ms

Figure 5: *Multi-view soft shadows rendered using traditional multi-pass rasterization of polygons (top) compared to VIR paired with our parallel view rendering approach (bottom). Both methods produce 128 high resolution depth maps and similar high quality shadow penumbra, but VIR takes only a fraction of the time.*

6. Remaining Challenges and Conclusion

Our remaining challenges center around improving control of sampling density, enabling the maintenance of high image quality *and* speed, especially as multiple views differ significantly:

Sub-pixel projected polygons might not cover a pixel center in the VIR raster sampling grid. In this scenario, VIR generates no samples for the polygon, even though in one or more of the rendered views, it *does* cover a pixel center. Important surface polygons might not be rendered. A conservative solution to this problem generates at least one sample for all polygons regardless of projected area. This transforms an undersampling problem into an oversampling problem. We implement this by enabling conservative rasterization (if available) during VIR or by storing the polygon's centroid instead of rasterizing.

Large and skewed projected polygons span a wide range of depth within a view. As a result, different areas on the polygon's surface may require very different sampling densities to maintain rendering efficiency. We might procedurally subdivide the polygon before VIR, reducing oversampling at the cost of additional subdivision logic. While VIR is flexible enough to address a few of these polygons, traditional view-dependent rasterization is purpose-built to render large projected polygons, and will be faster when they are common.

Polygons that vary greatly in projected size across many views require greatly varying sampling densities. We might address this problem by creating discrete point cloud LoDs of varying sampling density. This process replicates the input polygon a limited number of times (five replications or fewer performs well), and executes VIR using various distance values. This is illustrated in Figure 6. During rendering, the distance from the current view's eyepoint to a bounding volume containing the polygon is computed and used to determine the appropriate discrete point cloud to be used.

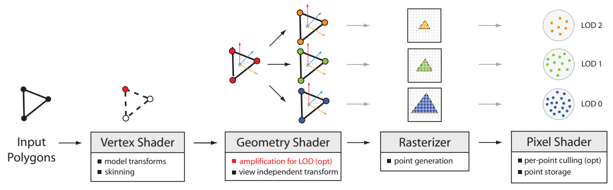

Figure 6: *Multiple discrete point-based LoDs computed using VIR.*

An alternative strategy with the potential to address all of these problems might adjust polygonal sampling density "just in time" during point generation, point rendering, or both. Inspired by [KV05], we might use a comparison of the point region's projected area to the area covered by a pixel, and use this to stochastically discard or generate polygon samples.

In conclusion, preliminary results show that *View Independent Rasterization* for multi-view rendering is a viable research direction warranting further investigation. Our future plans include testing this approach for more challenging multi-view effects including reflections, defocus blur, and diffuse global illumination using geometry representative of cutting-edge real-time game titles.

References

[Bær05] BÆRENTZEN J. A.: Hardware-Accelerated Point Generation and Rendering of Point-Based Impostors. *Journal of Graphics, GPU, and Game Tools 10*, 2 (2005), 1–12. URL: http://dx.doi.org/10.1080/2151237X.2005.10129197, doi:10.1080/2151237X.2005.10129197. 2

[BBH13] BARÁK T., BITTNER J., HAVRAN V.: Temporally Coherent Adaptive Sampling for Imperfect Shadow Maps. In *Proceedings of the Eurographics Symposium on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2013), EGSR '13, Eurographics Association, pp. 87–96. URL: http://dx.doi.org/10.1111/cgf.12154, doi:10.1111/cgf.12154. 2

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential Point Trees. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 657–662. URL: http://doi.acm.org/10.1145/1201775.882321, doi:10.1145/1201775.882321. 2

[GD98] GROSSMAN J. P., DALLY W. J.: Point sample rendering. In *In Rendering Techniques âĂŹ98* (1998), Springer, pp. 181–192. 2

[GP07] GROSS M., PFISTER H.: *Point-Based Graphics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. 1, 2

[Gre86] GREENE N.: Environment Mapping and Other Applications of World Projections. *IEEE Comput. Graph. Appl. 6*, 11 (Nov. 1986), 21–29. URL: http://dx.doi.org/10.1109/MCG.1986.276658, doi:10.1109/MCG.1986.276658. 1

[HA90] HAEBERLI P., AKELEY K.: The Accumulation Buffer: Hardware Support for High-quality Rendering. *SIGGRAPH Comput. Graph. 24*, 4 (Sept. 1990), 309–318. URL: http://doi.acm.org/10.1145/97880.97913, doi:10.1145/97880.97913. 1

[Hak01] HAKURA, ZIYAD S. AND SNYDER, JOHN M. AND LENGYEL, JEROME E.: Parameterized environment maps. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics* (New York, NY, USA, 2001), I3D '01, ACM, pp. 203–208. URL: http://doi.acm.org/10.1145/364338.364402, doi:10.1145/364338.364402. 1

[HREB11] HOLLÃĎNDER M., RITSCHEL T., EISEMANN E., BOUBEKEUR T.: ManyLoDs: Parallel Many-View Level-of-Detail Selection for Real-Time Global Illumination. *Computer Graphics Forum (Proc. EGSR 2011) 30*, 4 (2011), 1233–1240. 2

[Jon95] JONES M. W.: *3D Distance from a Point to a Triangle*. Tech. rep., Department of Computer Science, University of Wales, 1995. 2

[Kaj86] KAJIYA J. T.: The Rendering Equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 143–150. URL: http://doi.acm.org/10.1145/15922.15902, doi:10.1145/15922.15902. 1

[KV05] KALAIAH A., VARSHNEY A.: Statistical geometry representation for efficient transmission and rendering. *ACM Trans. Graph. 24*, 2 (Apr. 2005), 348–373. URL: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/1061347.1061356, doi:10.1145/1061347.1061356. 4

[LW85] LEVOY M., WHITTED T.: The use of points as a display primitive. *Technical report, Computer Science Department, University of North Carolina at Chapel Hill* (Jan. 1985). 1

[MKC07] MARROQUIM R., KRAUS M., CAVALCANTI P. R.: Efficient Point-Based Rendering Using Image Reconstruction. In *Eurographics Symposium on Point-Based Graphics* (2007), Botsch M., Pajarola R., Chen B., Zwicker M., (Eds.), The Eurographics Association. doi:10.2312/SPBG/SPBG07/101-108. 2

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 343–352. URL: http://dx.doi.org/10.1145/344779.344940, doi:10.1145/344779.344940. 1