High Performance Graphics (2016)
Ulf Assarsson and Warren Hunt (Editors)

Comparison of Projection Methods for Rendering Virtual Reality

Robert Toth[†], Jim Nilsson, Tomas Akenine-Möller

Intel Corporation

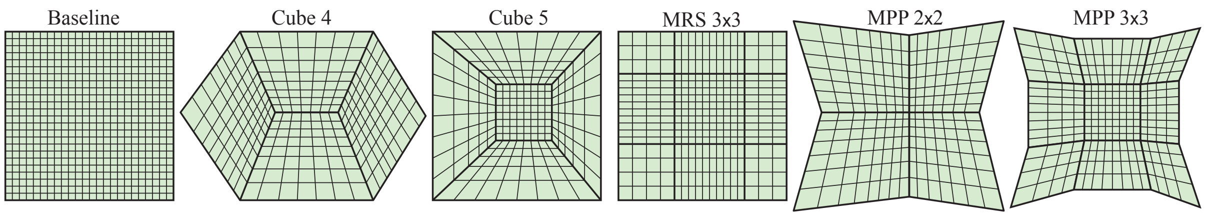

Figure 1: *The different types of projections that we evaluate in this paper. From left to right:* BASELINE*,* CUBE 4*,* CUBE 5*,* MRS 3×3*,* MPP 2×2*, and* MPP 3×3*, where* MRS *is multi-resolution shading and* MPP *is multi-plane projection. Note that the total resolution per projection is not comparable against other projections (merely to illustrate type of projection).*

Abstract

Virtual reality is rapidly gaining popularity, and may soon become a common way of viewing 3D environments. While stereo rendering has been performed on consumer grade graphics processors for a while now, the new wave of virtual reality display devices have two properties that typical applications have not needed to consider before. Pixels no longer appear on regular grids and the displays subtend a wide field-of-view. In this paper, we evaluate several techniques designed to efficiently render for head-mounted displays with such properties. We show that the amount of rendered pixels can be reduced down to 36% without compromising visual fidelity compared to traditional rendering, by rendering multiple optimized sub-projections.

1. Introduction

For as long as graphic displays have been around, they have mostly been regular grids of pixels, and with good reason. Regular structures are easy to manufacture, easy to index, and easy to render to. In addition, many man-years have been spent on optimizing rasterization on regular structures, so this process is very fast. Regular grids of pixels have served well as the fundamental image structure for both 2D and 3D graphics, and continue to do so even today. As versatile as regular grids may be, they come with one fairly significant limitation, namely, that they are quite inefficient at representing images spanning a wide field-of-view. This inefficiency stems from two factors. First, the periphery of the display is farther away from the observer compared to the display center, resulting in increasing angular pixel density towards the periphery. Second, the periphery is viewed at an angle, causing perspective foreshortening and further increasing angular pixel density and anisotropy towards the periphery. For a display spanning 90° field-of-view, the angular pixel density is 2.8 times as high at the edge as at the center.

Until recently, this has not been an important issue for most people, as we tend to use displays in a manner such that they rarely subtend more than about 45° of our visual field (consider your TV, computer display, or a typical cinema screen). At such small angles, the edge-to-center angular pixel density ratio is low, a mere 1.3 at 45°.

New immersive technologies are changing the way we view virtual environments. In particular, modern virtual reality head mounted displays (HMDs) present images spanning a much wider field-of-view than traditional displays. These HMDs are still based on regular grid displays, located close to the wearer's eyes. For the user to be able to clearly see the displayed image, the display surface is viewed through a lens system, making the display appear at a distance of several meters. These very same lenses serve another purpose, namely to counteract the poor angular pixel density properties of the regular grid display. By using lenses with pincushion distortion, the angular pixel density can be adjusted such that an undeservedly large fraction of the pixels do not end up in the periphery. This makes modern HMDs different from typical displays in a fundamental way as pixels do not appear on a flat, regular grid.

As mentioned above, rasterization-based graphics hardware

[†] e-mail: robert.m.toth@intel.com

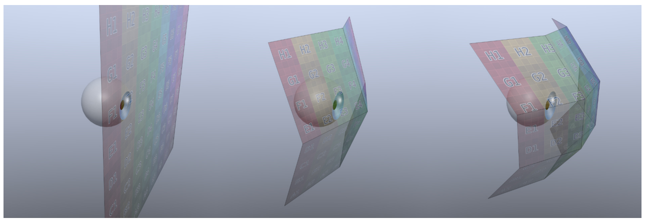

Figure 2: *In this paper, we study and evaluate several projection methods for more efficient VR rendering. Left: standard rendering for one eye using a single plane. Middle: multi-plane projection using 4 planes. Right: multi-plane projection with 9 planes.*

work on regular grids for good reasons. The incremental cost of rasterizing a pixel on a regular grid is only three additions and testing of three bits [Pin88]. Furthermore, the reciprocal w-component is a linear function over the grid, which allows for efficient computation and storage of depth and other interpolated values. Computing bounds on primitives is as trivial as computing the minimum and maximum of projected vertex coordinates. Due to these factors and others, uniform grids will likely remain the sample pattern on which rasterization hardware operate (subsamples can have arbitrary constant offsets relative to the uniformly spaced pixels, without foregoing the listed benefits.) To bridge the mismatch between uniformly rasterized grids and the non-uniform appearance of display pixels, the rasterized image is distorted in a final processing stage before being displayed. Emerging real-time ray tracing hardware may cast rays directly along the apparent direction of each pixel, but until such hardware matures, we are bound to continue rendering and distorting regular grids.

While the distortion stage can compensate for the mismatch between rasterized grids and the displayed pixel distribution, this comes at a cost. The grid's resolution must be adapted to the worst-case mapping density, lest the distorted result lose sharpness. Therefore, at highly distorted areas, many pixels may need to be rendered for each displayed pixel. The mismatch between rasterization and display can be reduced by using multiple sub-projections to produce a piecewise regular grid approximation of the non-linear display distribution. Depending on the distortion function, this can drastically reduce the required number of rendered pixels.

In this paper, we evaluate several ways of arranging sub-projections and optimize these as to minimize the required number of rendered pixels. See Figure 1. We also evaluate visual quality implications both for static images and in animation. Next, we present an overview of the algorithms that we use in this paper, and as such, the next section also serves as a review of relevant work.

2. Overview of Algorithms

In this section, we describe different algorithms to render images used as sources for the subsequent distortion computation. We begin by describing the most relevant algorithms which we have evaluated, and conclude this section by giving a brief overview of other algorithms which we have chosen not to include in our evaluation. Figure 1 visualizes some of the algorithms we investigate.

2.1. Basic perspective projection

Emerging VR APIs are at the time of writing still in beta stages, but none seem to provide information aimed at allowing applications to efficiently render using multiple projections.

We denote the simplest choice of rendering a single image (per eye) as BASELINE, and it will serve as a reference against which the other methods can be compared.

2.2. Cube projection

Thirty years ago, Greene and Heckbert [GH86] faced the problem of wide-angle non-linear pixel distributions when rendering for Omnimax cinema. In order to produce the 180° fisheye-distorted images required by the Omnimax projector, they saw two options. They could either modify all of their rendering tools to render directly in the required non-linear space, or let their unmodified tools render sides of a cube map [Gre86] and use a separate processing stage to combine and distort them into the required projection. They chose the latter approach and introduced the elliptically weighted average (EWA) filter to achieve high quality distorted images at a reasonable computational cost.

We evaluate two versions of partial cube mapping, namely, the four-sided layout proposed by Greene and Heckbert [GH86] for Omnimax projection and a five-sided layout missing only the rear-facing side. We refer to these as CUBE 4 and CUBE 5, respectively.

2.3. Multi-Res Shading

NVIDIA introduced multi-resolution shading (MRS) as part of their Gameworks VR suite [NVI15]. A single projection plane is divided into a grid of $M \times N$ co-planar sub-projections. The resolution of each sub-projection can be controlled individually, and so it is a significant improvement over BASELINE. Unlike CUBE 4 and CUBE 5, it cannot be used for projections exceeding 180°.

MRS can be implemented using any $M \times N$ number of sub-projections, though symmetry favours odd numbers for M and N. We refrain from using an excessive number of sub-projections and concentrate on evaluating only MRS 3×3, which is the number of planes used in the MRS programming guide.

2.4. Multi-Plane Projection

To reduce the pixel density mismatch between the rasterized image and the final display image, we present a novel technique which we call *multi-plane projection* (MPP). The nonlinear projection is synthesized from $M \times N$ connected linear sub-projections. Like MRS, the required resolution of each sub-projection can be determined individually. Further, the projection planes of each row and column of sub-projections can be tilted, stretched, and sheared, while preserving the original connectivity. This allows a great degree of freedom in the exact shape of sub-projections. In particular, MRS is a subset of possible MPP configurations. Two examples, MPP 2×2 and MPP 3×3, are illustrated in Figure 2. We limit our evaluation to MPP 2×2, MPP 3×2, and MPP 3×3 in order to keep the number of sub-projections to render within reason.

Lorenz and Döllner [LD09] rendered architectural visualizations

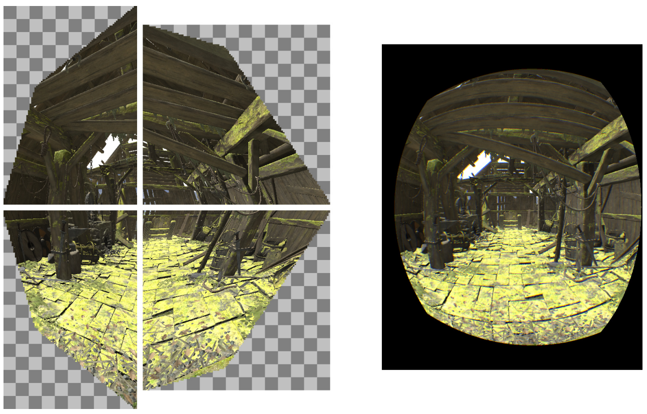

Figure 3: *Left: the four sub-projections of* MPP 2×2 *are shown with masking enabled. Parts that are not rendered are here shown as a gray checkerboard pattern. Right: the distorted result is unaffected by masking, since the masked area is never sampled.*

to be projected onto cylindrical surfaces. To avoid the blurring from resampling a cube map, they divided the cylindrical projection into 160 slices and approximated each by a perspective projection. They used this *piecewise perspective projection* (PPP) as an alternative to a post-process distortion. The authors tried applying PPP to a generic camera distortion consisting of 32,258 triangular perspective projections, but achieved performance far inferior to image-based techniques. While there are similarities between their work and MPP, there is a major difference in that they sought to *replace* image-based distortion processing, necessitating a large number of sub-projections optimized for direct visualization. Instead, we *optimize* image-based distortion processing by adapting the source images to better suit the subsequent distortion.

2.5. Variations

CUBE, MPP, and MRS all use multiple sub-projections to generate the full projection. The resolution required for each sub-projection can be determined individually. This can potentially cause a visible resolution discontinuity where sub-projections meet, if resultions are lowered for performance reasons. We therefore evaluate two variants of these algorithms – *unconstrained*, where the resolution of each sub-projection is determined without considering neighboring sub-projections, and *constrained*, where the resolutions are constrained such that neighboring sub-projections will have an equal number of pixels along shared edges, using the highest resolution of the connected sub-projections. The constrained variants thus require as high resolution as the unconstrained variants at best, and higher resolution in general.

Some regions of the sub-projections will not contribute to the final warped projection. Rendering to those regions can be prevented by using a stencil buffer [Vla15] or priming the depth buffer, thus preventing pixel shading from taking place. We refer to this as *masking*. An example of masking can be seen in Figure 3. Masking reduces shading computations and texture bandwidth usage, but it still requires triangle setup, rasterization, and depth/stencil testing to take place before the masked pixels can be discarded.

2.6. Other projection techniques

There are other techniques to either directly perform non-linear projection or otherwise approximate the required pixel density for later distortion.

Vertex shaders can be used to compute non-linear projections at vertices [BAS02, LGMM07]. While this method is feasible for finely tessellated objects, it is unsuitable for triangles spanning more than a few pixels as interpolation will not match the desired projection. This can cause various issues, such as visually deformed geometry and erroneous depth resolution.

With the advent of geometry shaders, bounding shapes of complex projections can be rasterized, leaving per-sample visibility testing to the pixel shader [TL08, MESL10]. Unfortunately, this class of algorithms has some significant performance implications. There is a computational overhead in performing per-sample visibility tests in the pixel shader, and a large amount of data needs to be written from the geometry shader for each triangle. Interpolation of depth and other attributes must be performed in the pixel shader. Early depth testing [MS04] is either disabled (for purely feed-forward systems) or impaired (for systems including feedback.) Planar surfaces no longer have planar depth equations either, so compression also suffers [Mor00,HAM06]. For the reasons listed above, we deem methods of this class infeasible for rendering complex scenes.

Various modifications to hardware rasterization have been proposed. As discussed in Section 1, non-linear methods [GHFP08] lack the efficiency of linear perspective projections. Coarse pixel shading (CPS) [VST*14, HGF14] has recently been proposed as a hardware-friendly way of reducing pixel shading, and can be applied to the periphery of an image to more closely resemble the desired pixel distribution. CPS does not alter actual pixel density, and so does not reduce rasterization cost, nor color and depth bandwidth usage. While the pixel density shape is more general than MRS, local densities are limited to powers of two, which limits its usefulness for our use case.

Guenter et al. [GFD*12] developed a system for foveated rendering using eye tracking. Their method uses overlapping cascades of varying size and resolution, with small high-resolution cascades being composited on top of large low-resolution cascades. A similar method using cascades could be employed for our purposes as well. We choose not to evaluate an adaptation of foveated rendering as it shares characteristics with MRS but is more constrained and would have significant overlap between cascades, necessitating heavy masking.

Finally, while real-time ray tracing may make all of this obsolete one day, it has yet to prove itself capable of matching or exceeding rasterization-based systems for real-time rendering.

We have chosen to focus on CUBE, MRS, and MPP. They can all be implemented on any existing graphics hardware, do not suffer from image artifacts, and are reasonably easy to integrate into applications.

Listing 1: *Pseudocode for stochastic optimization. As the temperature slowly decreases over many iterations, the Perturb function makes smaller and smaller parameter changes on average.*

```
function Optimize(iterations, temperature, cooldown):
    best_params = initial_parameters
    best_cost = EvaluateCost(best_params)
    while iterations > 0:
        new_params = Perturb(best_params, temperature)
        new_cost = EvaluateCost(new_params)
        if new_cost < best_cost:
            best_params = new_params
            best_cost = new_cost
        temperature = temperature * cooldown
        iterations = iterations - 1
    return best_params, best_cost
```

3. Optimization

Each algorithm has a parameter space with some number of degrees of freedom, which we denote k. The parameter space can include angles at which projection planes are split, tilting and shearing of each sub-projection, and so forth. For a full description of the various parameter spaces, see Appendix A. We seek the parameter vector $\hat{\mathbf{p}} \in \mathbb{R}^k$ that minimizes some cost function $C : \mathbb{R}^k \to \mathbb{R}$,

$$\hat{\mathbf{p}} = \underset{\mathbf{p}\in\mathbb{R}^k}{\arg\min}\, C(\mathbf{p}). \tag{1}$$

In the following, we will use the term *algorithm* to mean a particular configuration of either CUBE N, MRS $M\times N$, or MPP $M\times N$. Hence, as each algorithm has a pre-determined number of sub-projections, the cost of culling and issuing drawcalls should not be particularly sensitive to the exact set of parameters chosen. The per-pixel costs, on the other hand, are proportional to the sub-projections' resolutions, which in turn are highly dependent on the parameters. We therefore define the cost metric as

$$C(\mathbf{p}) = \sum_i C_i(\mathbf{p}) \tag{2}$$

$$C_i(\mathbf{p}) = w_i(\mathbf{p})h_i(\mathbf{p})m_i(\mathbf{p}), \tag{3}$$

where w_i and h_i are the pixel width and height of sub-projection i, and m_i is the masked ratio of the sub-projection. For non-masked variants, $m_i = 1$. While any optimization strategy can be used to find the globally optimal parameter vector $\hat{\mathbf{p}}$, we employ stochastic optimization [PTVF07]. Pseudocode for our implementation can be found in Listing 1.

There are many ways to define what resolution a given sub-projection should have. One could turn to Fourier analysis, under the assumption that the underlying signal is band-limited and will be perfectly reconstructed. Alternatively, one could base the resolution on how a hardware texture sampler would select mip levels when sampling the sub-projections to synthesize the final image. We have opted for a third option, which is to compute resolutions from cell spacings in local lattices as described below. Which method is preferable is subject to debate, but this is not the focus of our work. While we expect the results to vary slightly based on the chosen definition, this is a secondary effect and should not penalize any specific algorithm.

Pseudocode for how we opted to compute sub-projection resolutions is presented in Listing 2. Given a set of parameters $\mathbf{p}$, we first compute the sub-projection matrix $\mathbf{M}_i$ corresponding to those parameters. We then compute the clip-space vector along which each display pixel appears. We ignore all pixels falling outside the sub-projection's frustum, and compute the normalized device coordinates (NDCs) $\mathbf{n} \in [-1,1]^2$, for the remaining pixels. With x, y being display pixel coordinates, the partial derivatives $\partial\mathbf{n}/\partial x$, $\partial\mathbf{n}/\partial y$ are used to define a local lattice and a corresponding resolution, as illustrated in Figure 4. The lattice is computed per display pixel, so we need to combine all the per-pixel resolution requirements into one resolution to be used for the entire sub-projection. We simply use the 99th percentile of requested widths and heights. Our input vector fields are slightly noisy, and this choice prevents outlier data from causing resolutions higher than called for.

Listing 2: *Pseudo-code for determining the resolution of a sub-projection* `i` *with parameters* `p`*. This is executed once per sub-projection for each iteration of the stochastic optimization process.*

```
function LatticeResolution(dndx, dndy):
    if abs(dndx.y) > abs(dndy.y):
        sx = dndy.x - dndx.x * dndy.y / dndx.y
    else:
        sx = dndx.x - dndy.x * dndx.y / dndy.y
    if abs(dndx.x) > abs(dndy.x):
        sy = dndy.y - dndx.y * dndy.x / dndx.x
    else:
        sy = dndx.y - dndy.y * dndx.x / dndy.x
    return 2/abs(sx), 2/abs(sy)

function ViewFrustumCull(v):
    return abs(v.x) > v.w || abs(v.y) > v.w

function Percentile(list, percentile):
    sort_ascending(list)
    index = list.length * percentile
    return list[index]

function Resolution(p, i):
    M = SubProjectionMatrix(p, i)
    resolutions_x = empty list
    resolutions_y = empty list
    for each display pixel j:
        clip = M * view_vector[j]
        ndc[j] = ViewFrustumCull(clip) ? NaN : clip.xy / clip.w
    for each display pixel j:
        if ndc[j] is not NaN:
            dndx = ddx(ndc at j)
            dndy = ddy(ndc at j)
            rx, ry = LatticeResolution(dndx, dndy)
            resolutions_x.append(rx)
            resolutions_y.append(ry)
    res_x = Percentile(resolutions_x, 0.99)
    res_y = Percentile(resolutions_y, 0.99)
    return res_x, res_y
```

4. Results

We have optimized the parameters of each of the algorithms discussed in Section 2, using the process described in Section 3. We have optimized both with unconstrained resolutions, where each sub-projection can have any resolution, and with constrained resolutions, where adjacent sub-projections must have the same resolution along the shared edge (but not orthogonally to it). We also optimize for both unmasked and masked variants. Unless otherwise noted, presented results are unconstrained and are *not* employing masking. Where we *do* present results with masking, we assume zero cost of masked pixels. The true cost of masking therefore lies

Optimized costs, megapixels

Device	StarVR		Vive Pre		DK2		DK1	
Constrained	No	Yes	No	Yes	No	Yes	No	Yes
BASELINE	10.49 – 11.30	10.49 – 11.30	2.72 – 3.28	2.72 – 3.28	1.93 – 1.93	1.93 – 1.93	1.15 – 1.23	1.15 – 1.23
CUBE 4	6.18 – 9.56	6.22 – 10.67	1.74 – 2.27	1.78 – 2.31	1.46 – 1.88	1.46 – 1.90	0.61 – 0.79	0.61 – 0.79
CUBE 5	5.49 – 5.74	6.93 – 7.07	1.64 – 1.73	1.74 – 1.84	1.41 – 1.41	1.52 – 1.52	0.60 – 0.60	0.77 – 0.80
MRS 3×3	4.86 – 5.14	5.51 – 5.86	1.77 – 1.98	1.92 – 2.16	1.39 – 1.39	1.49 – 1.49	0.65 – 0.67	0.74 – 0.74
MPP 2×2	3.59 – 4.70	3.60 – 4.71	1.41 – 1.87	1.41 – 1.90	1.19 – 1.51	1.19 – 1.51	0.49 – 0.66	0.49 – 0.68
MPP 3×2	3.55 – 4.41	3.58 – 4.43	1.33 – 1.68	1.34 – 1.71	1.16 – 1.37	1.16 – 1.38	0.46 – 0.59	0.46 – 0.60
MPP 3×3	3.40 – 4.03	3.64 – 4.27	1.25 – 1.47	1.29 – 1.57	1.12 – 1.24	1.13 – 1.28	0.43 – 0.49	0.44 – 0.52

Table 1: *Amount of pixels that need to be rendered per eye for each algorithm. Constrained configurations use matching resolutions along edges between sub-projections. The lower bound of the ranges employ masking, while the upper bound does not. Masked configurations do not count peripheral pixels not contributing to the distorted image. All devices benefit from using multiple sub-projections, but the advantage is largest for devices with wide field-of-view, such as StarVR. Of the tested algorithms,* MPP 3×3 *consistently outperforms the others.*

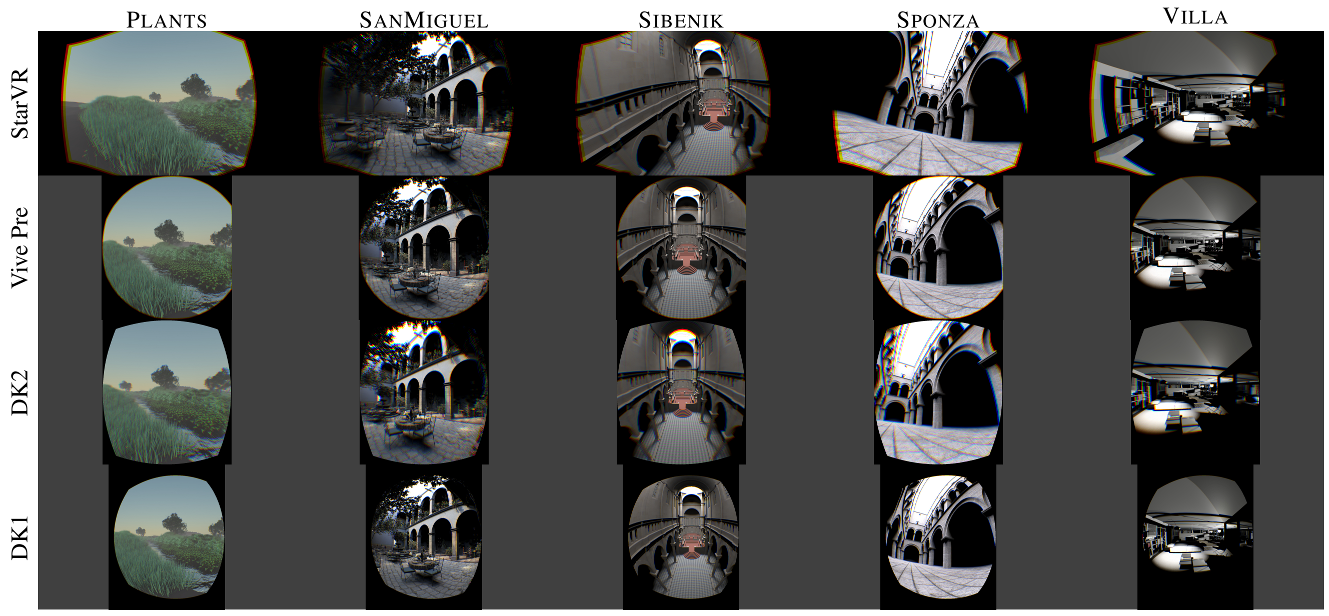

Figure 6: *Left eye view of five scenes from PBRT, rendered for three HMDs. We evaluate the different algorithms using these scenes to demonstrate that the algorithms are equivalent in terms of visual quality when using high-quality distortion. Rendered images for all scenes, algorithms and most HMDs are available as supplemental material.*

Image quality compared to ground truth

Scene	PLANTS	SANMIGUEL	SIBENIK	SPONZA	VILLA
BASELINE	42.4 ± 1.6dB	31.4 ± 1.0dB	40.3 ± 2.4dB	34.2 ± 1.3dB	31.9 ± 1.6dB
CUBE 4	42.2 ± 2.1dB	30.8 ± 2.5dB	39.1 ± 2.2dB	33.9 ± 2.0dB	32.9 ± 2.0dB
CUBE 5	41.7 ± 1.9dB	30.4 ± 1.5dB	39.5 ± 2.7dB	33.6 ± 1.7dB	31.6 ± 1.8dB
MRS 3×3	41.6 ± 1.9dB	30.6 ± 1.4dB	39.0 ± 2.3dB	33.8 ± 1.6dB	31.7 ± 1.7dB
MPP 2×2	41.0 ± 1.0dB	29.8 ± 1.0dB	38.3 ± 1.8dB	32.9 ± 1.4dB	31.0 ± 1.5dB
MPP 3×2	41.0 ± 1.1dB	29.7 ± 1.2dB	38.5 ± 2.0dB	33.2 ± 1.3dB	31.1 ± 1.5dB
MPP 3×3	40.9 ± 1.3dB	29.6 ± 1.3dB	38.4 ± 2.1dB	33.1 ± 1.5dB	31.0 ± 1.5dB

Table 2: *For each scene, the PSNR of each algorithm as compared to a high quality reference ray traced directly in distorted space. Each data point shows the range of PSNR for three of the HMDs (StarVR, DK1, DK2). As can be seen, the choice of algorithm has a rather insignificant impact on quality. The small variations most likely primarily stem from resolution differences (c.f. Table 1).*

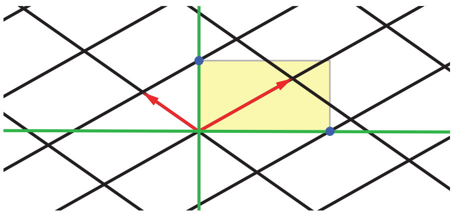

Figure 4: *Given two NDC vectors (red), we construct a lattice (black). Each axis of the lattice periodically crosses the green x- and y-axis. Pixel spacing is chosen based on the first intersections (blue circles) along both the x-and y-axis. The sub-projection resolution follows from the required pixel spacing (yellow rectangle). This corresponds to lines 1–10 in Listing 2.*

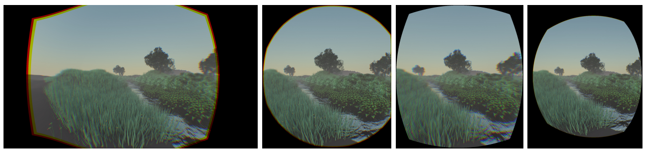

Figure 5: *A scene rendered for the left eye of the four devices used for our results. From left to right: Starbreeze StarVR* (2560×1440)*, HTC Vive Pre* (1080×1200)*, Oculus DK2* (960×1080)*, and Oculus DK1* (640×800)*.*

somewhere between the reported non-masked and masked results, depending on the application (see Section 2.5 for details.) We have performed all of the optimizations using four different lens distortions: an Oculus DK2 using its 'A'-lenses, providing 100° field-of-view in 1.04 MP per eye with low distortion; an Oculus DK1 using its 'A'-lenses, providing 110° field-of-view in 0.51 MP per eye with more distortion; an HTC Vive Pre, providing 110° field-of-view in 1.30 MP per eye with similar distortion; and a Starbreeze StarVR, prodiving 210° field-of-view in 3.69 MP per eye with the largest distortion of the four devices. The four distortions are shown in Figure 5. For each algorithm, the total number of pixels summed over each sub-projection is listed in Table 1.

To validate the correctness and image quality of each algorithm, we have rendered several scenes with each of them using PBRT (extended with more flexible camera models.) These can be seen in Figure 6. Reference images were produced by directly ray tracing the distorted image, using a per-pixel distortion map containing camera-space view vectors. A large number of rays were used (256 rays/pixel) to eliminate Monte-Carlo noise. For the remaining images, each sub-projection was first ray-traced to an image with a large number of rays (256 rays/pixel) to eliminate Monte-Carlo noise. The resulting images were then texture mapped onto planes corresponding to each sub-projection, and these were subsequently ray traced using the reference method at 64 rays/pixel to produce the distorted images. Table 2 shows, for each scene, PSNR of each algorithm as compared to the reference rendering. The quality of warping from sub-projection to the final image will be limited by real-time constraints. We have therefore included a comparison of how the different algorithms fare when only a single bilinear texture lookup is used per display pixel in Figure 7.

Batch count and render time

Device	StarVR	Vive Pre	DK2	DK1
BASELINE	231 (9.4 ms)	215 (1.2 ms)	207 (1.2 ms)	213 (1.2 ms)
CUBE 4	433 (9.6 ms)	426 (2.6 ms)	410 (2.7 ms)	440 (2.8 ms)
CUBE 5	450 (8.3 ms)	432 (2.7 ms)	424 (2.7 ms)	431 (2.7 ms)
MRS 3×3	666 (10.4 ms)	636 (4.3 ms)	617 (4.3 ms)	620 (4.2 ms)
MPP 2×2	390 (6.5 ms)	395 (2.4 ms)	373 (2.3 ms)	424 (2.5 ms)
MPP 3×2	477 (7.1 ms)	508 (3.2 ms)	488 (3.1 ms)	527 (3.4 ms)
MPP 3×3	630 (8.5 ms)	674 (4.4 ms)	631 (4.3 ms)	654 (4.3 ms)

Table 3: *Number of batches issued by Unity 5.3 to render the scene shown in Figure 9, as well as corresponding render times for our user-space implementation.*

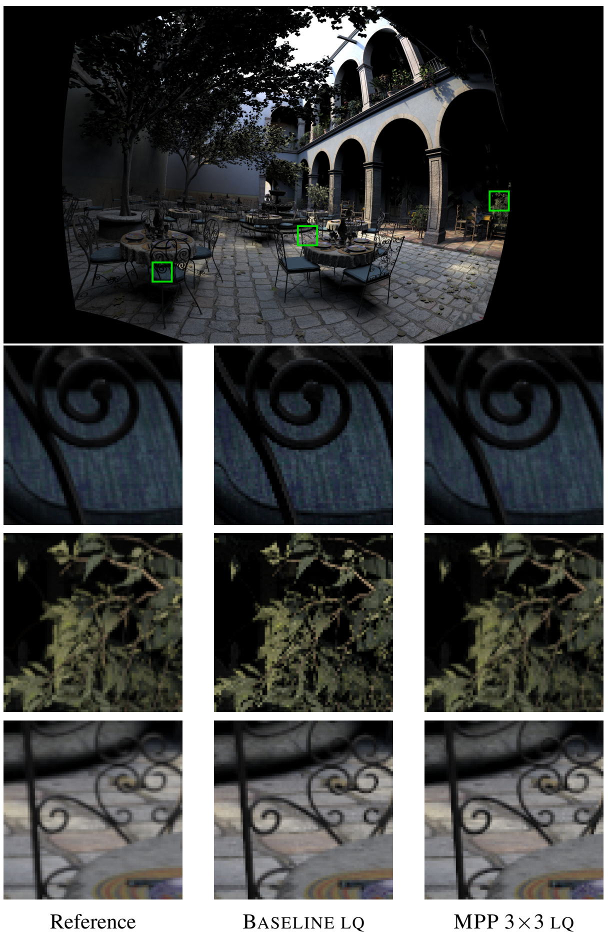

Figure 7: *Impact of using multiple sub-projections instead of one when trading distortion performance for quality. The reference distorts the single* BASELINE *sub-projection with 64 EWA filtered samples using PBRT.* BASELINE LQ *and* MPP 3×3 LQ *are distorted using a single bilinear sample per pixel. Heavy minification in the periphery with* BASELINE LQ *causes aliasing.* MPP 3×3 LQ *has less minification and therefore exhibit no visible aliasing. Near the image center, minification is modest for both algorithms, resulting in similar quality for both.*

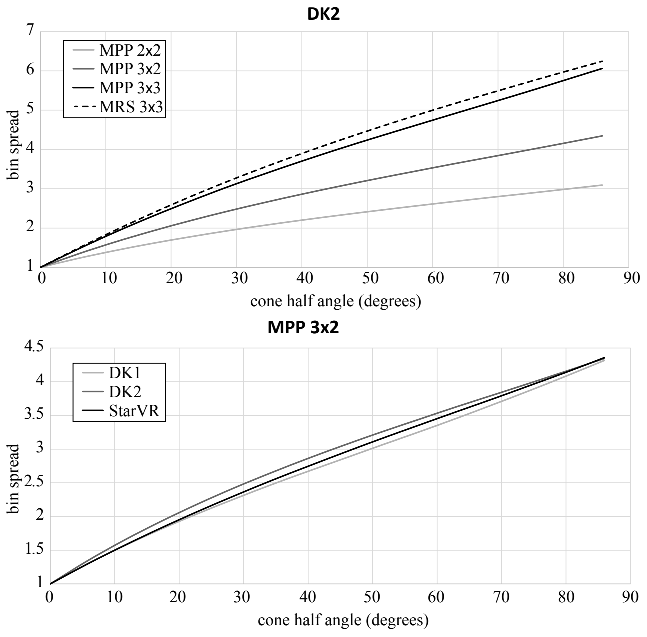

Figure 8: *In these diagrams, we show bin spread (overlap) over sub-projections as a function of the half-angle of a cone. Top: bin spread for* MPP *and* MRS *when rendering for the DK2. Bottom: bin spread for* MPP 3×2 *when rendering for DK1, DK2, and StarVR. The simulated cones represent object bounds used for culling purposes. At one extreme, an object with small extents is unlikely to straddle the border between sub-projections, thus having an average bin spread near* 1*. At the other end, objects subtending a large angle overlap many of the sub-projections on average.*

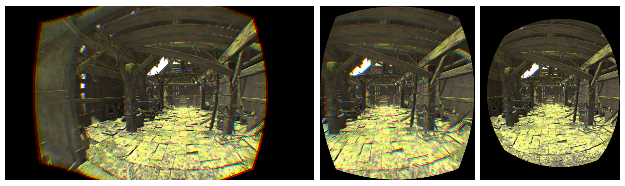

Figure 9: *The Blacksmith scene by Unity. We have created multi-camera setups in Unity 5.3 for each algorithm, and measured the number of batches issued by Unity to render them all, as well as render time (see Table 3.) From left to right: StarVR, DK2, & DK1.*

When rendering to multiple sub-projections, drawcalls of a scene should not be re-submitted to the GPU once for each sub-projection. Instead, view-frustum culling should be performed for each sub-projection in order to issue drawcalls only to the sub-projections which the geometry overlaps. Figure 8 shows a chart of expected overlap, also called *bin spread* [CSI*98], as a function of object size. We numerically simulate bin spread for the optimized configurations by randomizing cone directions and computing cone-frustum intersections. We vary cone half-angles between zero and 90° (i.e., a solid angle of 2π sr,) and disregard from any cone not intersecting any sub-projection frustum so that we simulate only objects that need to be rendered.

To see the effect using a professional game engine and high quality assets, we have created user-space camera configurations in Unity 5.3 for all of the techniques, and rendered the Blacksmith scene shown in Figure 9. The number of drawcalls used for the various algorithms is listed in Table 3. The render times on our system (Intel Core i7-3960X, NVIDIA GTX 970) are also included.

5. Discussion

The authors believe immersive display technology will continue to increase in field-of-view, and that it is time to rethink the way we use rasterization to render for such displays. While most first-generation consumer-grade VR equipment will likely have characteristics close to those of DK2 or Vive Pre, we believe this will rapidly shift to be more in line with StarVR, and then continue beyond what is achievable today. The exact nature of angular pixel distribution remains an open question, but it seems clear that the single planar projection (BASELINE) used for traditional displays is ill-suited for this type of immersive technology. In the long term, ray tracing may make emerging display pixel distributions a non-issue, but for now, rasterization remains the only viable option for high quality real-time graphics in immersive environments. Luckily, rasterization can be used in different ways than originally envisioned, and we have therefore investigated several ways in which to make rasterization better suited for this new task.

The base premise of most techniques we have looked at is that by rendering several sub-projections instead of just one, each projection can be made significantly less expensive by locally lowering the resolution without compromising visual quality. We have showed in Table 2 that all methods presented meet high quality standards for the final images, and Figure 7 shows that most of them work better than BASELINE even when distortion processing is implemented with performance, rather than quality, in mind. It is worth noting that any image quality issues in rendering the sub-projections themselves will propagate to the distorted end result. Geometric aliasing may make the underlying planar structure noticeable as the orientation of staircase artifacts will change at sub-projection borders. While a full user study is outside the scope of this work, subjective evaluation by the authors did not reveal any visible artifacts at sub-projection edges using the optimized parameters. When lowering the resolution by 2× in each direction, such a change in aliasing pattern could be noticed, but the authors found the effect to be insignificant in comparison to the aliasing itself, which under those circumstances was highly objectionable. Theoretically, if the distortion processing exhibits sharpness patterns due to aliasing when sampling sub-projection texels, those patterns may abruptly change characteristics where sub-projections meet. Again, subjective evaluation by the authors did not reveal any sharpness patterns, neither at sub-projection edges nor elsewhere. We therefore recommend using non-constrained, non-planar methods, as these provide the largest savings, with no visible downsides. Constrained or co-planar methods should only be used if application-specific algorithms require some aspect thereof, such as each edge pixel having a single well-defined neighbor in the adjacent sub-projection.

Absolute performance is, as always, highly dependent on the content being rendered. It also depends on how optimized the rendering engine is for the techniques being employed. We have therefore focused on two key metrics, namely, the total number of pixels spanned by all sub-projections, and the statistically expected number of sub-projections overlapped by objects. The first metric should correlate well with per-pixel work, such as rasterization, pixel shading, texture bandwidth usage, and so on. The second metric should correlate well with CPU overhead for dispatching drawcalls and GPU time for state changes and vertex processing.

Looking at the pixel metric in Table 1, we make several observations. Using multiple sub-projections is highly beneficial, with up to 64% reduction of pixels compared to BASELINE. The difference depends on the amount of distortion, and highly distorted devices have more to gain by using multiple sub-projections. CUBE 5 often falls between MPP 2×2 and MPP 3×2, as might be expected from the number of sub-projections, but its topology requires opposing sub-projections to be parallel to each other, which limits its flexibility and causes it to perform rather poorly in some cases. MPP 2×2 is better than MRS 3×3 with high distortion, and vice versa with low distortion, so the choice seems to depend on the devices being targeted. As expected, MPP 3×3 is superior to MRS 3×3 with an equal amount of sub-projections due to the additional degrees of freedom it offers.

With regards to the overlap metric, Figure 8 shows that using a set of N sub-projections does not increase bin spread by N. Many objects are contained entirely within a single sub-projection and do not incur significantly more processing overhead than if we were to render only a single projection. This is also evident in Table 3, where we implement the algorithms in Unity 5.3 to verify this theory. For 4, 5, 6, and 9 views, we get average bin spreads of 1.8, 2.0, 2.3, and 2.9, respectively. It is worth noting that the Unity game engine is unaware of the coherent nature of the sub-projections, and is rendering one camera to completion at a time.

With only simple pixel shading and a naïve implementation of rendering multiple sub-projections, the overhead of rendering objects several times outweigh the savings from reduced pixel processing. This can be seen in Table 3, where only StarVR (the highest-resolution device) has enough pixels for this to be the dominating cost. Should the algorithms be implemented and optimized for in the engine, performance per sub-projection would likely improve. Modern graphics APIs such as Direct3D12, Vulkan, and Metal, have been designed to dramatically reduce the cost of issuing many drawcalls, further making the CPU overhead less of an issue. There are many ways in which multiple projections can be rendered on modern graphics hardware, some of which we list here.

1. One projection can be rendered to completion at a time. This is what our proof-of-concept implementation in Unity does.
2. Geometry shaders can replicate primitives to sub-projections.
3. Instancing can be used to do the same using extensions (e.g., `GL_AMD_VERTEX_SHADER_VIEWPORT_INDEX` which is supported by all major vendors), typically with better performance than using a geometry shader.
4. MRS was most probably specifically designed to leverage the `GL_NV_viewport_array2` extension, with which MRS can be rendered in a single pass by careful configuration of viewports and scissor regions.
5. A recent extension, `GL_OVR_multiview`, allows all algorithms we have evaluated to be rendered in a single pass.

The resolution requirements we present assume the target pixel density to match the display pixel density. Targeting a higher density to increase sharpness is possible even in the presence of multiple sub-projections, by simply scaling the resolution of each. Similarly, dynamically scaling resolution to achieve performance goals is also possible, though magnification during distortion may reveal the underlying plane structure if scaling down too agressively.

Regardless of chosen algorithm, the use of multiple sub-projections makes image-space processing techniques (e.g., bloom, defocus and motion blur, SSAO) more difficult to perform pre-distortion, as filter footprints may extend into adjacent sub-projections. Using the constrained variants may make such techniques easier to implement, but still more complex than for BASELINE. It is worth noting that many image-space effect implementations are uniform over a projection, thus assuming a reasonably small field-of-view. This may translate poorly to VR unless they are adapted to take projection properties into account such that they filter over the desired solid angle.

We used the devices' distortion to define our required sub-projection resolution. It is conceivable to use other targets as well, e.g., an ad-hoc importance map or based on the human visual system. However, optimizing the parameters of an algorithm takes considerable time, so we do not believe these algorithms should be used for eye-tracking purposes unless parameters are pre-computed and tabulated in some way. It might be better to combine one of the presented algorithms with foveated rendering [GFD*12] or coarse pixel shading [VST*14] to efficiently render both the foveal region and the periphery.

Overall, we feel that MPP 2×2 should meet the needs of applications in the immediate future, and that MPP 3×3 might be a good choice in a few years when consumer devices with wider field-of-view become widely available. Any application implementing MPP should be able to support various layouts and choose a suitable one depending on the connected HMD and on GPU performance characteristics.

6. Future work

We believe eye tracking cameras will be an essential component in future HMDs due to their versatility. They can be used for automatic calibration, adjust for pupil location dependent lens distortion, interaction in virtual worlds, establishing eye contact in social applications, and accelerate rendering. We therefore see rendering for devices with wide field-of-view in the presence of eye tracking as an important avenue of future work, exploring the best way of combining the distortion-based wide field-of-view algorithms presented herein with foveated rendering based algorithms.

A user study for finding subjectively acceptable ad-hoc resolution target maps for different HMDs would be useful in further reducing the computational cost than what our objective resolution target allows. Ideally, HMD vendors should provide APIs to query such data (or directly provide sub-projection configurations) based on such user studies conducted for each specific device.

Efficient distortion-aware image-based techniques need to be developed for special effects we have come to expect from modern 3D graphics, such as bloom, defocus blur and motion blur. Filter kernels that have been separable when rendering for traditional displays may no longer be so due to spatially varying filter footprints, and so new algorithms may be required to tackle these issues again. These algorithms also need to be adapted to seamlessly filter over multiple sub-projections.

Acknowledgements

We would like to thank Emmanuel Marquez and Lionel Anton at Starbreeze for providing us with distortion data for StarVR. We would also like to thank Alex Vlachos at Valve for providing us with a Vive Pre, and Warren Hunt at Oculus for providing us with DK2s. Finally, we would like to thank David Blythe and Charles Lingle at Intel for supporting this research.

References

[BAS02] Brabec S., Annen T., Seidel H.-P.: Shadow Mapping for Hemispherical and Omnidirectional Light Sources. In *Computer Graphics International* (2002), pp. 397–408. 3

[CSI*98] Chen M., Stoll G., Igehy H., Proudfoot K., Hanrahan P.: Simple Models of the Impact of Overlap in Bucket Rendering. In *Graphics Hardware* (1998), pp. 105–112. 7

[GFD*12] Guenter B., Finch M., Drucker S., Tan D., Snyder J.: Foveated 3D Graphics. *ACM Transactions on Graphics, 31*, 6 (2012), 164:1–164:10. 3, 8

[GH86] Greene N., Heckbert P.: Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter. *IEEE Computer Graphics and Applications, 6*, 6 (June 1986), 21–27. 2

[GHFP08] Gascuel J.-D., Holzschuch N., Fournier G., Péroche B.: Fast Non-Linear Projections using Graphics Hardware. In *ACM Symposium on Interactive 3D Graphics and Games* (2008), pp. 107–114. 3

[Gre86] Greene N.: Environment Mapping and Other Applications of World Projections. *Computer Graphics and Applications, IEEE 6*, 11 (November 1986), 21–29. 2

[HAM06] Hasselgren J., Akenine-Möller T.: Efficient Depth Buffer Compression. In *Graphics Hardware* (2006), pp. 103–110. 3

[HGF14] He Y., Gu Y., Fatahalian K.: Extending the Graphics Pipeline with Adaptive, Multi-Rate Shading. *ACM Transactions on Graphics, 3*, 4 (2014), 142:1–142:12. 3

[LD09] Lorenz H., Döllner J.: Real-time Piecewise Perspective Projections. In *International Conference on Computer Graphics Theory and Applications* (2009), pp. 147–155. 2

[LGMM07] Lloyd D. B., Govindaraju N. K., Molnar S. E., Manocha D.: Practical Logarithmic Rasterization for Low-error Shadow Maps. In *Graphics Hardware* (2007), pp. 17–24. 3

[MESL10] McGuire M., Enderton E., Shirley P., Luebke D.: Real-Time Stochastic Rasterization on Conventional GPU Architectures. In *High-Performance Graphics* (2010), pp. 173–182. 3

[Mor00] Morein S.: ATI Radeon HyperZ Technology. In *Graphics Hardware, Hot3D Proceedings* (2000). 3

[MS04] Mitchell J. L., Sander P. V.: Applications of Explicit Early-Z Culling. In *Real-Time Shading (ACM SIGGRAPH Courses)* (2004). 3

[NVI15] NVIDIA: Multi-Res Shading. Programming Guide, November 2015. PG-07866-001_v01. 2

[Pin88] Pineda J.: A Parallel Algorithm for Polygon Rasterization. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (1988), vol. 22, ACM, pp. 17–20. 2

[PTVF07] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007. 4

[TL08] Toth R., Linder E.: *Stochastic Depth of Field using Hardware Accellerated Rasterization.* Master's thesis, Lund University, 2008. 3

[Vla15] Vlachos A.: Advanced VR Rendering. Game Developer's Conference (presentation), 2015. 3

[VST*14] Vaidyanathan K., Salvi M., Toth R., Foley T., Akenine-Möller T., Nilsson J., Munkberg J., Hasselgren J., Sugihara M., Clarberg P., Janczak T., Lefohn A.: Coarse Pixel Shading. In *High-Performance Graphics* (2014), pp. 9–18. 3, 8

Appendix A: Parameter spaces

The three HMDs we have optimized for are all symmetric around the horizontal plane. We have therefore omitted some degrees of freedom (DOF) which would produce assymetric configurations. We rely on per-pixel view vectors as input. Below, we use a coordinate convention in which x is right, y is up, and z is back.

Baseline only has a single DOF, namely the projection plane normal's azimuth. The projection plane extents are given by the extents of the view vectors' intersections with the projection plane.

Cube 4 has four DOF. One edge lies on the xz-plane, along some angle. The top and bottom planes joined by this edge are separated by some aperture angle. The shared edge's endpoints define where the left and right planes are located.

Cube 5 can be seen as Baseline extruded by some depth, and therefore has two DOF.

MRS 3×3 has five DOF: the Baseline azimuth, and the top, bottom, left, and right fractions where the projection is split to form sub-projections.

MPP $M\times N$ has $2M+\lceil\frac{3}{2}(N-1)\rceil-2$ DOF. There are $M-2$ azimuthal angles at which the xz-plane is split, in addition to the two endpoints which are given by the view vectors. Each of the M columns can also be tilted to some angle in the xz-plane. Each of the $\lfloor N/2\rfloor$ sub-projection rows above the horizontal plane can be sheared both in the xy- and yz-planes. Finally, there are $\lfloor(N-1)/2\rfloor$ rows with adjustable heights, not counting the topmost (and bottommost) rows which are given by the view vectors.

There are two additional degrees of freedom we have omitted, in addition to the symmetry-breaking ones mentioned above, as both Cube 4 and Cube 5 could be sheared in the xz-plane.