Efficient Rendering for Light Field Displays using Tailored Projective Mappings

LAURA FINK, SVENJA STROBEL, LINUS FRANKE, and
MARC STAMMINGER, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

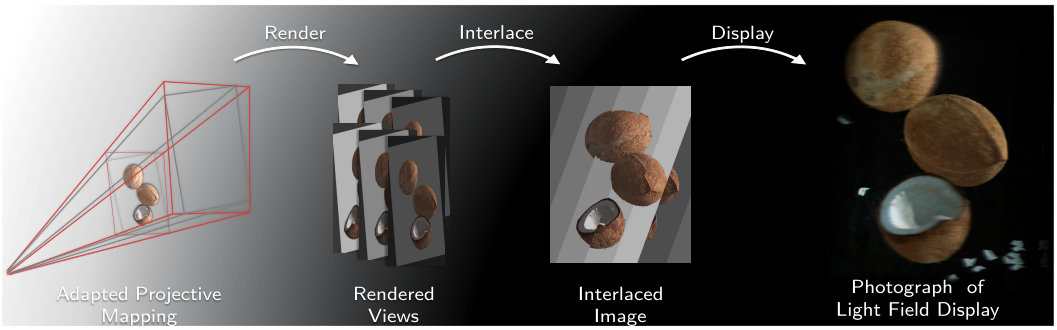


Fig. 1. Adaptive projective mapping for efficient rendering delivered by light field displays. We augment the projection matrix to reduce wasteful pixel computations, saving up to 80% of the memory consumption needed in simple setups and increasing performance by up to 7×.

A standard paradigm when rendering for parallax-based light field displays is to render multiple, slightly offset views first and to interweave these afterwards. In practice, more than 40 views of preferably high resolution need to be rendered per frame to achieve acceptable visual quality. The total amount of rendered pixels may consequently exceed the native resolution of the display by far. Increased memory consumption and sub-optimal render times are direct consequences.

In this paper, we examine where pixels are "wasted" and present novel projective mappings for the virtual camera system that are custom tailored to such displays. Thus, we alleviate the aforementioned issues and show significant performance improvements regarding render time and memory consumption, while having only minor impact on visual quality. As we mainly touch the projective mapping of the virtual camera, our method is lean and can easily be integrated in existing rendering pipelines with minimal side effects.

CCS Concepts: • **Computing methodologies** → **Rasterization**; *Antialiasing*; • **Hardware** → **Displays and images**.

Additional Key Words and Phrases: Rendering, Light Field Display, Projection Matrix, Super Sampling

ACM Reference Format:
Laura Fink, Svenja Strobel, Linus Franke, and, Marc Stamminger. 2023. Efficient Rendering for Light Field Displays using Tailored Projective Mappings. *Proc. ACM Comput. Graph. Interact. Tech.* 6, 1 (May 2023), 17 pages. https://doi.org/10.1145/3585498

Authors' address: Laura Fink, laura.fink@fau.de; Svenja Strobel, svenja.strobel@fau.de; Linus Franke, linus.franke@fau.de; Marc Stamminger, marc.stamminger@fau.de, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstr. 11, Erlangen, BY, Germany, 91058.

2577-6193/2023/5-ART $15.00
https://doi.org/10.1145/3585498

1 INTRODUCTION

Light field displays deliver three-dimensional (3D) content to an observer's eye without the need of any glasses. They simulate the experience of looking out of a window by coupling pixels not only to a two-dimensional position on the screen but also a direction. Hence, distinct views can be displayed for two eyes and also accommodate different head positions, which is a property also known as *multiscopy*.

Various display technologies were developed to implement such optical behavior. Some achieve multiscopy by layering multiple LCD attenuaters [Wetzstein et al. 2011] over a strong backlight. Also multi-projector setups can be used to achieve the desired effect [Ma et al. 2019]. Parallax-based methods trade spatial resolution in favor of angular resolution by hiding some pixels of a conventional 2D screen depending on the view direction [Ma et al. 2019]. For the 3D effect of the Nintendo 3DS [Stuart 2011], parallax-barriers were used to mask out pixels that should be non-visible for a specific view direction. Integral imaging relies on an array of tiny lenses that redirect light emitted by pixels to achieve multiscopy. Lenticular lenses work in a similar way but are of cylindrical form. These are used for flip images [Ma et al. 2019]. Lenticular lenses only allow for a horizontal light field, so the observed view depends on the horizontal position of the observer only and the 3D effect disappears when the eyes align with the vertical axis of the display.

Naturally, rendering for such devices becomes very challenging, as a 3D position in the virtual scene could be observed from multiple angles simultaneously, and visibility and appearance have to be evaluated for each view distinctly. To be compatible with standard rasterization or ray tracing pipelines, a common paradigm is to first render multiple, slightly offset views and then interweave them together into a format that is tailored for the specific display technology [Shen et al. 2022]. While the upper limit of the angular resolution is defined by the capabilities of the display optics naturally, the number of rendered views fused is crucial for a smooth blending from one view to the next when e.g the user moves their head.

The methods presented in this paper mainly target the aforementioned parallax-based display types and were implemented and evaluated on the Looking Glass Portrait [2022b]. This type of display has lenticular lenses incorporated[1]. The manufacturer recommends to render more than 40 views per frame with reasonable resolutions [Looking Glass 2022c], for i.e. Looking Glass Portrait the recommended configuration is 49 views at $420\,px \times 560\,px$ per view. That is more than *five times* the number pixels compared to its actual native display resolution of $1536\,px \times 2048\,px$. In order to increase the apparent visual fidelity, it is sometimes recommended to use an even higher resolution per view at the cost of even higher computational costs and memory consumption [Looking Glass 2022a]. In practice, this may lead to extreme oversampling as way more pixels might be rendered than actually displayed on the screen.

In the course of this paper, we analyse how such interweaving is implemented and based on that show that not all pixels of the source images are used in the target image. The methods presented in this paper aim to prevent such oversampling by only adapting the perspective mappings used during rendering. In summary this is our contribution:

- We analyse how the source views are interweaved, resulting in the *interlaced image* that is natively displayed on the screen to identify where pixels are overdrawn.
- We present an adaptation to the perspective mapping that is compatible with standard rasterization pipelines reducing pixel-load on real-world hardware by up to 80 percent.

[1]The manufacturer has not published specifics on the used optics. We derived this based on observations of the display (rendering, no 3D effect when display is rotated by 90 degrees and no prominent view-dependent changes in brightness, which would be typical for parallax-barriers) and rendering (2D render target).

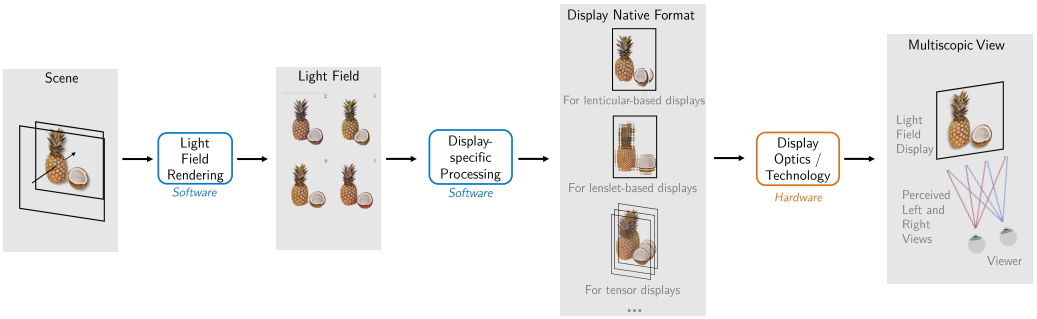


Fig. 2. Standard image formation paradigm for light field displays. A fronto-parallel light field is rendered from a 3D scene. The output light field representation is resampled or processed into a format that can be natively displayed on the specific technology. When observed on the hardware e.g. through specific optics, attenuation layers, or because of specialized projection, the image is perceived as 3D.

- We show a super sampling extension of our mapping that can efficiently and easily be realized to reduce aliasing.

We show that significantly lower render times and memory consumption can be achieved at the cost of acceptable visual differences. An implementation is available on https://github.com/lorafib/projs-for-lfd-rendering.

2 RELATED WORK

In this section, we provide an overview of other rendering algorithms targeting light field displays but also have a short glance on light fields in general. We start with the naive way to render for such displays. Then, we discuss work regarding the plenoptic function and light fields for an overview of the underlying theoretical concepts detached from technological specifics. Afterwards, we detail works that specifically target light field displays. Thereby, we cover algorithms that are based on the foundations of light fields (usually their inherent redundancies) first, and then discuss miscellaneous approaches.

Rendering for Light Field Displays

In the most general (and naive) way, rendering for most light field displays boils down to rendering a fronto-parallel light field. To do so, commonly a camera system is set up that comprises multiple sub views. For each of these views, images are rendered one after another. Then this light field is translated into a format that can be natively displayed by the respective technology. Figure 2 illustrates this procedure. At the beginning of Section 3, we detail on this paradigm for the case of lenticular-based horizontal light field displays which are the focus of this work.

Note that there is a recent survey by Shen et al. [2022] on rendering for light field displays which also provides an overview of the various display technologies.

Efficient Light Field Rendering

The *plenoptic function* [Bergen and Adelson 1991] is a high-dimensional function that yields the light intensity when evaluated for any wavelength at any three dimensional position coming from any direction at any point in time. Thus, it describes any light that potentially could have been sensed. This theoretical vehicle is used to formulate *light fields* [Levoy 2006] in computer graphics

and vision. As light fields are discretized subsections of the universal plenoptic function, rendering from them essentially boils down to evaluating a more or less complex look-up-table. Light fields are parameterized according to underlying geometric primitives, for instance spheres [Fink et al. 2019; Todt et al. 2008] or other arbitrary surfaces [Oechsle et al. 2020]. Another type is the fronto-parallel lightfield [Gortler et al. 1996; McMillan and Bishop 1995] where the parameterization is based on two parallel planes. In principle, a light field display can be understood as a device that is capable to directly display such a four dimensional light field, whereas a conventional display only shows a two dimensional slice of the plenoptic function.

Because of this theoretic relation, rendering for light field displays can benefit from any algorithm that enables fast generation of fronto-parallel light fields provided that the re-sampling to the devices native format (or the interface format) can be done efficiently. Recent work in that regard is e.g. based on *multi plane images* [Flynn et al. 2019; Wizadwongsa et al. 2021] which allow for very fast rendering of views within the field of view of the static light field. Others encode light fields in implicit representations which can also encode some seconds of the time domain [Bemana et al. 2020; Suhail et al. 2022]. However, these methods focus more on the (time-consuming) implicit scene reconstruction and do not evaluate the generation of multi plane images from synthetic scenes.

Use cases that demand scene interaction and real-time framerates exclude all methods that need long pre-processing times to generate some complex data structure first, no matter how fast the rendering from it might be afterwards. In practice, rendering even small scenes puts high load on the hardware just because of the sheer amount of pixels that need to be processed and transferred per frame. Specialization to the targeted display optics and hardware of the rendering algorithms is thus particularly advisable to reduce the amount of data that is generated and processed per frame to the absolute necessary.

Specialized Rendering for Light Field Displays

Various approaches try to make use of the redundancy in light fields by proposing sparsely filled light fields which intermediate views can be interpolated from. Guan et al. [2019] synthesize a dense light field from four corner views using depth-image-based rendering and a hole filling algorithm which uses the average pixel color of the preceding frame. By doing so, they achieved real-time performance when rendering for a lenslet-based display [Guan et al. 2019]. Similarly, Guo et al. [2022] predict a densely sampled light field from sparsely rendered key frame views. They warp source views into the target view using an estimated optical flow and cancel out color variations using a color calibration network. They achieve a framerate of over 20 fps for 60 dense views at a resolution of 1024×512 from 11 input views.

Synthesizing the native image format for tensor displays is an inverse problem. Wetzstein et al. [2012] show that this is possible to achieve in real-time by implementing the simultaneous algebraic reconstruction technique in an efficient manner. However, they do not go into detail where the original light field came from to begin with. Guan et al. [2020] exploit the coherence of rasterization calculations across views. To do so, they propose a hierarchical soft rendering pipeline enabled by a GPU-driven software rasterizer. Bettio et al. [2008] incorporate an adaptive multi resolution approach delivering high framerates even for large triangle meshes. In the domain of ray tracing, Pang et al. [2017] propose to directly render the native format of a lenticular display by tracing the view ray associated to a given pixel.

3 RENDERING FOR PARALLAX-BASED HORIZONTAL LIGHT FIELD DISPLAYS

In this section, we provide details on the display specific camera setup and processing step for lenticular lenses-based horizontal light field displays. Figure 3 provides an illustration of the steps

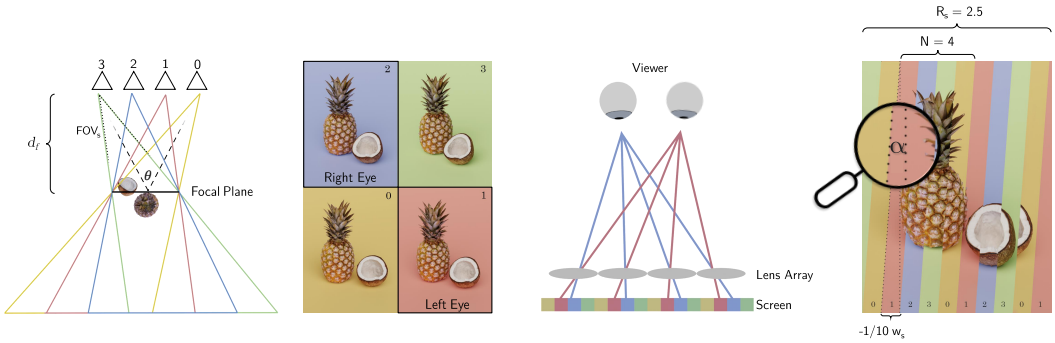


Fig. 3. Overview. (a) Example camera system with 4 views. d_f is the distance to the focal plane, FOV_s is the field of view of the display. (b) Images as seen from the 4 views. (c) Schematic layout of the display. A lens array deflects light emitted by the screen so that the observer sees the correct views. (d) Image that is directly displayed on the light field display screen. R_s is the number of repetitions of all views along the horizontal axis. On stripe will have a width of 0.1 w_s if the number of views $N = 4$ and the number of repetitions $R_s = 2.5$.

involved. While it is the most naive version to perform rendering for such displays, it is very versatile and is applicable for any type of light field display. As mentioned before, the standard paradigm comprises the following steps. First, a camera system is setup with multiple sub views (Figure 3 (a)) and for each of these views, an image is rendered one after another (Figure 3 (b)). An image that can be displayed natively by the display is interlaced from diagonal strips of the source views eventually (Figure 3 (d)). The strips align with the tilt of the lenticular lenses. A slight tilt of these lenses was found to eliminate moiré patterns [Li et al. 2015].

In the following, we go more into details of this paradigm and introduce necessary variables for the mathematical framework. We limit our considerations to horizontal parallax-based light field display but concepts of this section are transferable to all parallax-based light field displays.

Camera System

Each of N views is placed along a line and point in parallel directions. The frusta of these views are skewed so that they intersect at the focal plane. The screen's field of view (FOV_s) dictates the maximum distance between two views depending on the distance to the focal plane d_f.

Thus, we can calculate the View Matrix V_i of view i with

$$V_i = VT(t_i), \tag{1}$$

where $T(t_i)$ is the translation matrix of t_i (t_i is the last column of T) which is given by

$$t_i = V \begin{pmatrix} x_i \\ 0 \\ -d_f \\ 1 \end{pmatrix}; \qquad \text{with} \quad x_i = -d_f \tan\left(\left(\frac{i}{N-1} - 0.5\right)\theta\right), \tag{2}$$

where θ is the opening angle defining the spread of the outermost cameras. V is the base view matrix derived from the pose of the whole camera system which has its origin at the center of the focal plane.

We denote the rendered image width and height per view as w and h, and for the screen's native resolution we use w_s and h_s. Thus, the aspect ratio r_s is calculated by $r_s = w_s/h_s$. We can determine the view specific skew s_i by

$$s_i = \frac{x_i}{d_f \tan(\text{FOV}_s/2) r_s}. \tag{3}$$

Hence, the projection matrix of view i is

$$P_i = P(r_s, FOV_s, n, f, s_i), \tag{4}$$

derived from the standard projection matrix with horizontal skew:

$$P(r, \phi, n, f, s) = \begin{pmatrix} \frac{1}{r\tan(\phi)} & 0 & s & 0 \\ 0 & \frac{1}{\tan(\phi)} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \tag{5}$$

n and f denote the near and far values of the frustum.

Interlacing

Naturally, the specifics of this step are highly dependent on the incorporated lenses or parallax-barriers. But for parallax-based light field displays the general idea is always the same. After interweaving the pixels of the various views into a single 2D array of pixels, the display optics should deflect the emitted light so that the specific view is shown when looked at from the corresponding gaze angle. Figure 3 (d) provides an example of how the result of such a step can look like.

When working with horizontal light field displays we mainly consider the following parameters: α_s which is the angle of tilt of the image stripes that are interlaced, R_s which is the number of repetitions of all images along the horizontal axis and corresponds to the number of lenses. w_s and h_s describe the native screen resolution. Additionally, manufacturers may provide a calibrated position of the actual center or some display corner to allow for perfect alignment of optics and the interlaced views on the screen (e.g. the center texture coordinate C_s). Color channels may also be treated differently to account for the subpixel positions of the red, green and blue diode [Li et al. 2015].

The following snippet illustrates how such an interlacing shader can be implemented [Guan et al. 2020].

```
// inputs:  tc: texture coordinate of the output pixel,      N: number views,
//          C_s: calibrated center of the display,           R_s: number of
    reptitions,
//          tilt: tan(alpha_s)/r
// output:  pixel_color: color displayed on the screen at tc

vec3 rgb[3];
// Iterate over each color channel
for (int i=0; i < 3; i++) {
    // Calculate view index
    float cur_repetition = (tc.x + tc.y*tilt + i*color_offset) * R_s - C_s;
    float norm_index_01 = mod(cur_repetition+ ceil(abs(cur_repetition)), 1.0);
    int view_index = norm_index_01 * N;
    // Read color
    rgb[i] = readColor(view_index, tc);
}

// Combine to output color
pixel_color = vec4(rgb[0].r, rgb[1].g, rgb[2].b, 1);
```

Analysis of the Interlacing Sampling

We analyse the interlacing by projecting back the sampled pixels to the source views. This can be seen for example in Figure 4. In general, we notice that many pixels are unused, and that especially with higher resolutions the ratio of unused to used pixels dramatically shifts. While for lower resolutions, the access pattern appears to be without any clear rule, configurations with higher resolutions reveal the strips that samples follow along. With increasing resolution of the source images, the distance between these strips becomes bigger as well. A higher number of views does not affect the distance between the strips but affects the frequency of accesses along the sampling strip. This is due to the specifics of the display optics. In order to use every pixel of the rendered source views, the amount of pixels beneath one lenticular lens should match the number of views rendered. However, the amount of pixels beneath one lenticular lens is $w_s/R_s = 6.2\,\text{px}$ for the Looking Glass Portrait, while simultaneously more than 45 individual renderings are recommended. Even if sampling is done for each channel of the RGB diodes individually (thus 19 samples in total) many pixels are unused. Consequently, parts of the rendered views are not read during the interlacing causing an oversampling during rendering.

In the following sections, we make use of the observations made in this section for our novel perspective mapping scheme, where we cut down the number of wasted pixels dramatically.

4 ADAPTING THE PROJECTIVE MAPPING

In case of light field displays, subpar resolution renderings will result in an image with notable aliasing or blurriness. However, increasing the resolution is extremely costly in terms of memory and performance. In the preceding section (and Figure 4), we found that the image displayed on the screen is composed of many interlaced strips of the source views. Pixels that do not lie on that strip will never get displayed. Hence, simply increasing the resolution will mainly result in pixels that will not be displayed and only a fraction of additional samples will influence the interlaced image. Based on these observations, we introduce an adaptation to the projective mapping of the camera system that is tailored to match the sampling during interlacing more precisely, thus, allowing higher resolutions as computations are only done where necessary.

Figure 5 illustrates the idea schematically. The introduced projective mapping incorporates a rotation that matches the tilt angle α. As the image slices belonging to one view have uniform distance in the interlaced image, we set the horizontal resolution to match the number of repetitions. Because rasterization results are evaluated at pixel centers, a column of pixels in the source view matches a strip sampled during interlacing. The choice of image strip for each view is controlled via the skew. The areas in the top-left and bottom-right corner will not be used for the interlaced

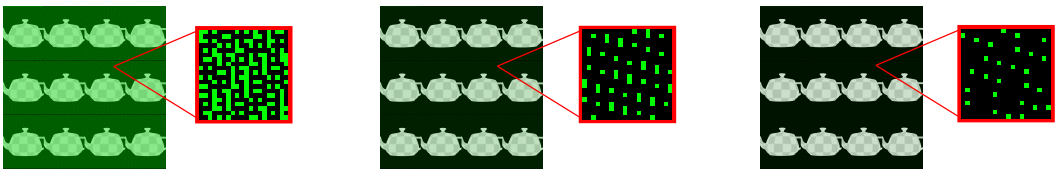


Fig. 4. Access patterns of interlacing steps for different settings of w, h and N (shown here is only the green channel). (a) 420×560 px per view, $N = 48$. (b) 768×1024 px per view, $N = 48$. (c) 768×1024 px per view, $N = 108$. In general, with higher resolutions more pixel computations are unnecessary in standard projection setups.

image, hence they are masked out during rendering. The projective mapping is adapted so that the aspect ratio fits the increased height and slimmer width.

Note that this approach only works in a reasonable way if $w_s/R_s \leq N$ is fulfilled. Otherwise the sampling strips during interlacing are wider than one pixel and thus we cannot approximate the outcome using our method sufficiently. Also note that hardware-accelerated anti-aliasing methods like MSAA are not trivially compatible with our approach as the subpixel offsets would be at areas we actually want to skip. At the end of this section, we introduce a super sampling method that is compatible with our approach. The adaptations introduced in the following are based on the rotation of the frustum according to the lens tilt angle. An alternative to this approach might be the use of a shearing transformation. The method described is only applicable to parallax-based horizontal light field displays as is, like lenticular lens-based or parallax barrier-based displays. The general idea should be extensible to lenslet-based displays though, see Section 7.

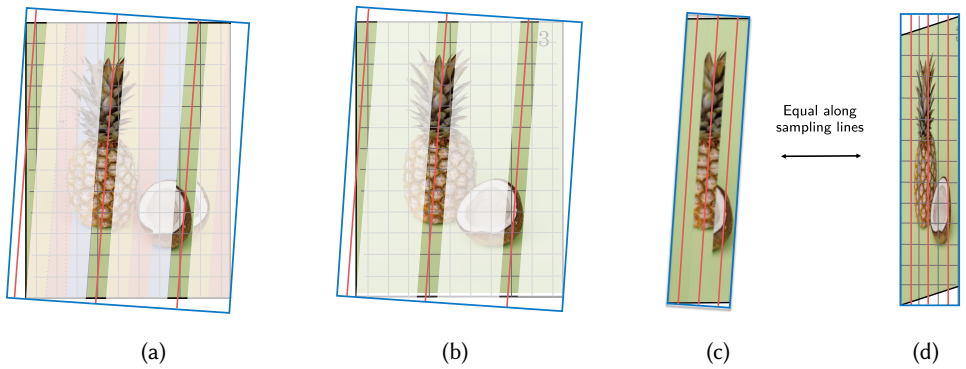


Fig. 5. Schematic derivation of the adapted projective mapping. The red lines indicate the sample positions of the interlacing step. The blue boxes show the edges of the adapted projective mapping. (a) The interlaced image, with sections masked out which do not originate from *View 3*. (b) Source view #3. Sections not used in the interlacing image are masked out. (c) View 3 sliced and condensed to required sections. (d) Schematic result after rasterization using the adapted projective mapping. The pixel centers are aligned with the sampling strips.

Adapting the Camera System

The view matrices as defined in Equation 1 are used as is.

We construct the new projection matrix by rotating the corners of the quad $\{(r, 1), (-r, 1), (-r, -1), (r, -1)\}$ by the tilt angle α around its center and determine the resulting axis-aligned bounding box; see Figure 6a. We make use of the width and height of the bounding box to calculate the new aspect ratio r'. Thereby, we need to enlarge r' to cover a full number of repetitions, as illustrated in Figure 6a.

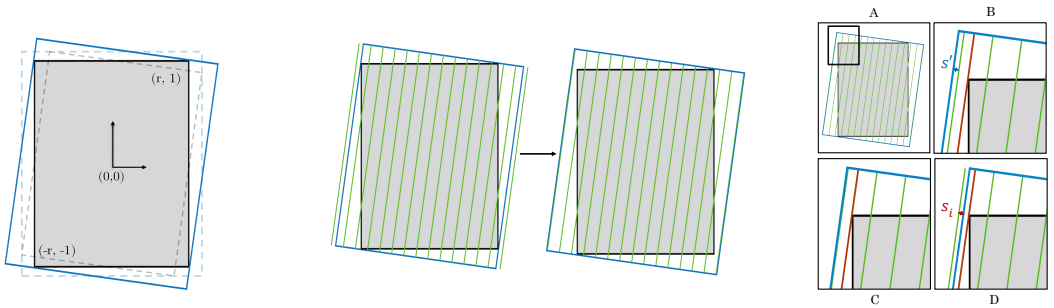
Thus, the adapted width w' is equal to the full number or repetitions covered by the enlarged bounding box. We set the new image height h' to the height of the bounding box of the rotated quad $\{(w/2, h/2), (-w/2, h/2), (-w/2, -h/2), (w/2, -h/2)\}$. Using Equation 5, we can calculate the adapted projection matrix P_i' by

$$P_i' = AP(1, FOV_s, n, f, s_i'), \quad A = S_x(1/r')R_z(\alpha)S_{xy}(h'/h) \tag{6}$$

where S and R are scale and rotation matrices, their subscripts indicate the affected dimensions. We calculate the adapted skew parameter s_i' by

$$s_i' = s' + s_i, \tag{7}$$

where s' aligns the left and right edges of the projected frustum planes according to the display calibration as show in Figure 6c. In practice, s' also contains the subpixel offset of the green channel (more details are given in Figure 8).



(a) (b) (c)

Fig. 6. Forming the adapted frustum of the projective mapping. (a) The blue dashed line indicates the bounding box of the quad with upper-right corner $(r, 1)$ and lower-left corner $(-r, -1)$ (solid black line) which was rotated around its center (black dashed line). Then the bounding box is rotated by α again. (b) In order to let the frustum cover a full number of image stripes, the aspect ratio needs to be adapted. (c) s' shown in box B is the skew offset that aligns the frustum with the interlacing strips according to the display calibration. The view dependent s_i, as known from Equation 3.

Adapting the Interlacing Step

In Section 3, the interlacing step is illustrated by a schematic code snippet. For the adapted version, the view index calculation remains unchanged. We only touch the view local texture coordinates in order to align with the adapted projective mapping. The following code summarizes the adaptations needed.

```
// inputs:  ...,
//          P_i[N]: P_is of all views, P_i_new: P_i's of all views  (see Eq.s 5
    and 6 )
// ...
// Iterate over each color channel ...
    // Calculate view index ...
    int view_index = ...
    mat3 P_i = mat3(P_i[view_index]); P_i[2][2] = 1;
    mat3 P_i_new = mat3(P_i_new[view_index]);  P_i_new[2][2] = 1;
    vec2 tc_new = (0.5*(P_i_new*inverse(P_i)*(vec3(2*tc-1,1))+0.5).xy;
    // Read color
    rgb[i] = readColor(view_index, tc_new);

// Combine to output color ...
```

Note that the texture look up done in `readColor()`, does a bilinear interpolation which we found to cause less aliasing even though this might mix up colors corresponding to different views.

Super Sampling

As introduced at the beginning of this section, rendering in higher resolutions becomes wasteful when using the reference paradigm. However, using the adapted projected mapping we can control sampling positions much more fine-grained than before, allowing us tailored anti-aliasing sampling patterns. Specifically, this is also important for our method, as the immense reduction of pixels along the width of the source images in some cases can lead to subtle aliasing, especially along axis-aligned straight lines.

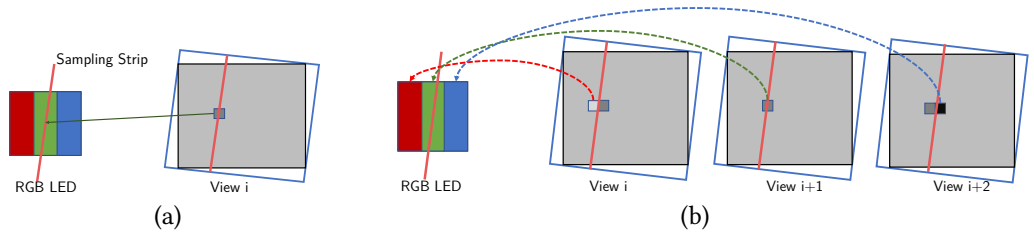


Fig. 7. Visualization of the sub-pixel accesses during the interlacing step with our adapted projective mapping from the previous section. Instead of using each color channel from a source pixel for the target display pixel (a), the view index is computed per-color channel individually to account for the subpixel offsets of the physical LEDs (b). In case of our standard adapted projective mapping, the sampling strip (red line) is aligned with the green center subpixel. This leads to a slight systemic incorrectness which is counteracted by the introduced $P'_{R,i}$ and $P'_{B,i}$, thus adding two additional sampling strips in order to match the sampling positions per color channel and make anti-aliasing easily accessible.

Looking at super sampling algorithms, the obvious goal is to spend samples where they have the biggest impact [Sherrod 2008]. A method which is an inspiration for our sampling adaptations is the subpixel-based anti-aliasing technique ClearType [Basit Ali et al. 2022]. Therefore, we recapitulate the interlacing accesses from the code snippet in Section 3 and take into account that the color channels of a pixel are at slightly offset positions on a RGB LED, thus requiring slightly different sampling positions in the indexed view. For a visualization, see Figure 7. Figure 8d shows that while we render most of the pixels accessed during interlacing, we miss out on some of the accesses corresponding to the red and blue channels. This is due to our alignment of pixel columns to the sampling strip of the green channel (which is centered between the other two), thus we miss out on red and blue sample positions specifically.

Using these findings, we propose a naive super sampling strategy whereby we render multiple times per view using different projection matrices. Thus, we allow for a super sampling individualized for each color channel, which maps perfectly to the hardware. To this end, we introduce two additional matrices $P'_{R,i}$ and $P'_{B,i}$ for the red and blue parts of the pixel. As we know that the single color diodes are offset horizontally, we only have to adapt the skew parameter. We define the three matrices by

$$P'_{R,i} = AP(1, FOV_s, n, f, s'_i - s_c), \qquad P'_{G,i} = P'_i = AP(1, FOV_s, n, f, s'_i + 0),$$
$$P'_{B,i} = AP(1, FOV_s, n, f, s'_i + s_c), \tag{8}$$

where s_c is the skew offset which we coupled to the color-dependent offset given by the display calibration. The intensity of the color channel of a pixel in the interlaced image is composed by the three different outcomes using $P'_{R,i}, P'_{G,i}$ and $P'_{B,i}$.

In Figures 8c and 8e, we set s_c to the doubled offset of the calibration which worked best empirically. Figure 8f shows the union of pixels rendered, while pixels not rendered are masked

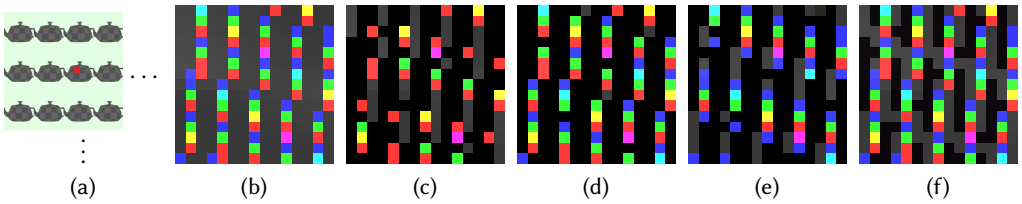


Fig. 8. Access pattern of interlacing step for a setting of $768\,px \times 1024\,px, N = 48$. (d)-(f) show the same close-ups but pixels not rendered by our method are masked black using either P'_Rs, P'_Gs, or P'_Bs. Thus, bright colors indicate pixels which were rendered and accessed and grey pixels indicate pixels which were rendered but not accessed. (a) Subset of the 48 views. (b) A close up showing the access pattern of the standard interlacing step as known from Figure 4. Same close up, pixels not rendered using (c) P'_R, (d) P' or (e) P'_B are masked out. Note that when comparing (b) and (c), (d) or (c), we miss out on some accesses for subpixels of other color channels. (f) Pixels not rendered using either P', P'_R or P'_B are masked out. Here, all accesses are covered.

out black. This approach is naturally transferable to super sampling in the temporal domain, but we leave this as future work.

5 RESULTS

In this section, we evaluate the performance of our proposed method in terms of memory, quality and computation time. First, we describe our experimental setup. Afterwards, we detail on memory savings, rendering performance and achieved quality.

Experimental Setup

We tested the implementation on three different scenes (Teapot, Buddha and Coconuts) with varying complexity, see Table 2. Our system was equipped with an Nvidia Geforce RTX 3060 GPU and an AMD Ryzen 5 5500 CPU. OpenGL was used as the framework's rasterization backbone. We implemented our method using the publicly available Looking Glass Portrait [Looking Glass 2022b]. Its calibration parameters are listed in Table 1. In total, we tested five settings with different numbers of views N (48, 108) and render resolution w and h. Table 3 provides an overview of these settings and introduces the abbreviations used. The lower bound for the w and h values (LR) were chosen based on the manufacturer's recommendation for real time applications [Looking Glass 2022c]. The upper bound for w and h (HR) is based on the native resolution of the screen. We did not evaluate the configuration high resolution with 108 views (HR, 108) as the default interface of the display expects all source views to be in a single texture. So, this setup would have exceeded the maximum texture size allowed by OpenGL.

α	-7.96 degree
FOV_s	14 degree
w_s	1536 px
h_s	2048 px
R_s	246.88 px
subpixel width	0.000217014 (in tc)
C_s	0.04347775 (in tc)

Table 1. Lightfield display parameters.

Scene	Vertices	Texture Memory
Teapot	8378	0.11 MB
Buddha	549333	4.53 MB
Coconuts	991879	170.78 MB

Table 2. Evaluation Scenes.

Setup Name	Low Resolution **LR**		Medium Resolution **MR**		High Resolution **HR**	
Resolution	Reference	Ours	Reference	Ours	Reference	Ours
	420×560 px	294×613 px	768×1024 px	441×1121 px	1526×2048 px	294×2243 px
# Views N	48, 108		48, 108		48	

Table 3. Resolution (LR, MR and HR) and number of view N configurations. Resolutions $w \times h$ indicated per view.

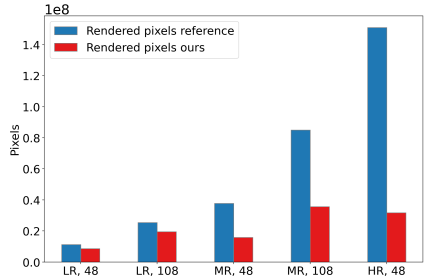


Fig. 9. Total number of pixels for various configurations. Numbers of "Rendered pixels ours" do not consider stencil masks.

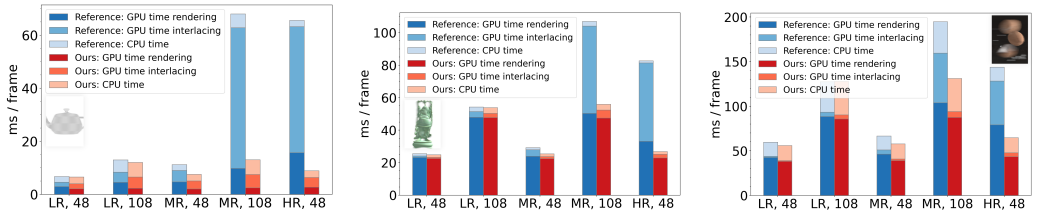


Fig. 10. Time measurements of our and reference renderings. Labels along the x-axes correspond to settings of "Resolution, N" as introduced in Table 3 . (a) Teapot. (b) Buddha. (c) Coconuts.

Experiments

Table 3 shows the difference in resolution needed for our versus the reference approach, as our approach consistently requires less pixels to be computed. This can also be seen in Figure 9, which shows the total number of pixels for all source images combined. While the number of pixels scales quadratically in case of the reference approach, we only scale linearly depending on the used height. Also, given the display parameters as listed in Table 1, we mask out approximately 22 % of pixels during rendering, as the corners of our source images exceed the viewport of the interlaced image.

We measured rendering performance in terms of ms per frame. To this end, we averaged frame times for a time span of 30 s while the objects in the scenes completed a full 360 degree rotation. The outcomes are presented in Figure 10. Our approach consistently outperforms the reference, up to a factor of 7.3× in the highest resolution setup with the teapot scene.

In regards of visual fidelity and general image quality, we performed a quantitative evaluation, shown in Figure 11, and a qualitative evaluation of representative views in Figure 12a, 12b and 12c. To this end, we mainly compared the pixel-wise differences (L1) of the interlaced images. In the

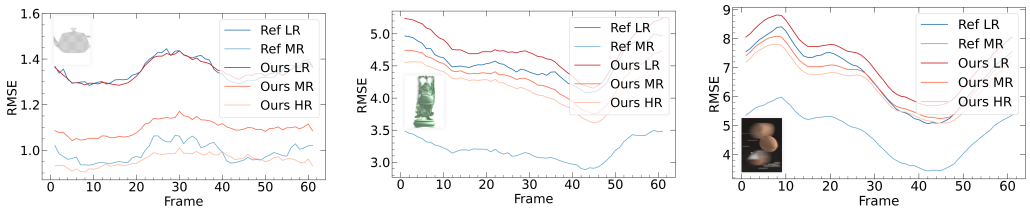


Fig. 11. RMSE of various configurations compared against the configuration (Ref., HR, 48) for a 360 degree rotation of the respective object. Pixel intensity was in the range [0,255]. (a) Teapot. (b) Buddha. (c) Coconuts.

quantitative analysis of Figure 11 we specifically compared all other configurations with $N = 48$ against the configuration (Ref, HR, 48) which we consider as ground truth for the remainder of this evaluation. We did not consider any perception-based error metrics as the interlaced image is obviously not meant to be perceived directly.

The results show that in setups with similar render times as our method, our method consistently shows better results. Meanwhile in other setups, there are differences dependant on the scene, with our approach quality-wise even outperforming the reference on the Teapot scene.

Additionally, see Figure 13 for photographs of the display. There we see slight differences at the edges, but otherwise correct pixel interlacing and high quality views creation.

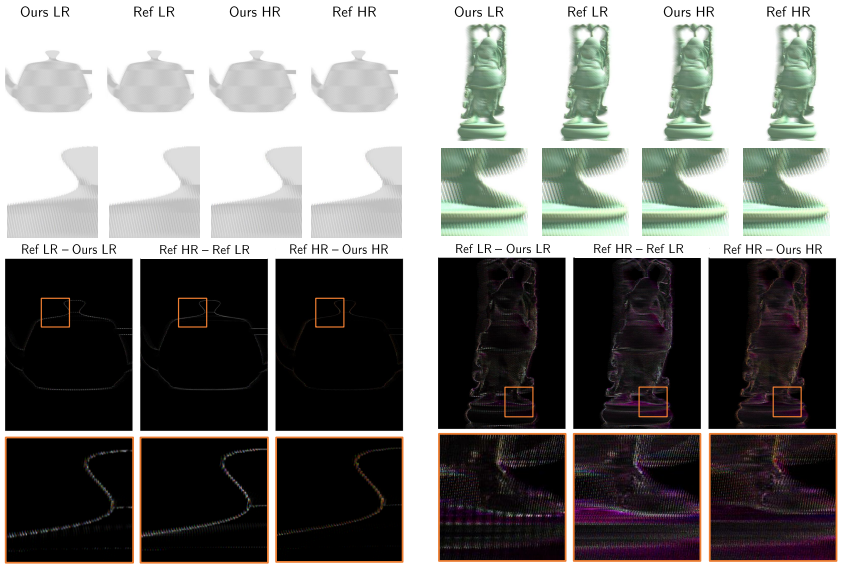
Super Sampling

Figure 14 and 15 summarize the results of the introduced super sampling approach. Super sampling allows the results to be considerably sharper compared to the standard version of our method and mitigate aliasing. Our naive implementation however adds significant render times, thus we recommend using a temporal setup for this. As seen in Figure 13 (bottom right), our super sampling approach effectively smooths out overestimated details.

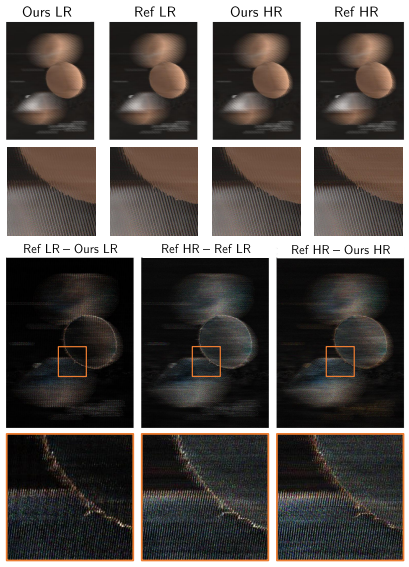
6 DISCUSSION

Considering the measurements from the previous section, we identify two configurations with immediate use cases. First, using the adapted projective mapping for the configuration (Ours, LR, 48) we consistently outperformed the reference counterpart (Ref, LR, 48) for all scenes in regards of rendering time and memory consumption. Performance gains of the Buddha and Coconut scenes were accompanied by a relatively higher error when comparing the RMSE of (Ref, LR, 48) and the ground truth (Ref, HR, 48) and the RMSE of (Ours, LR, 48) and the ground truth (Ref, HR, 48). In case of the Teapot, no increase in error was found. Second, we found that the configuration (Ours, HR, 48) shows promising results in case of Teapot scene, slightly surpassing (Ref, MR, 48) in terms of RMSE using less memory and similar performance. However, these results did not translate to the Buddha and Coconut scenes which exhibit more high frequent details and textures.

In general, we find that the memory and time consumption increases linearly with a very low slope for configurations of higher resolution with constant N using the adapted projective mapping. Similarly, image quality expressed as RMSE decreases only with a lower slope compared to the reference counter parts. This is mainly due to configuration changes only affecting the height of the source view image while the width is kept constant. Additionally, we introduced a systematic error where we disadvantage samples for red and blue subpixels because of the alignment of the pixel centers with the green sample positions.



(a) Teapot. (b) Buddha.



(c) Coconuts.

Fig. 12. Exemplary interlaced images and difference visualizations for various configurations and scenes. For (a), (b), and (c) respectively: First row shows the interlaced image of the configurations (Ours, LR, 48), (Ref, LR, 48), (Ours, HR, 48) (Ref, HR, 48). Second, row shows close ups of these interlaced images. Third row shows the absolute pixel-wise difference between different configurations as indicated by the captions. The difference is multiplied by 10 for better visualization.

Thus, we see the presented super sampling method as extremely promising to overcome this trend and improve quality with less pixels rendered. Our naive implementation was not able to be competitive in terms of render time by any means. Still, we find further quality improvements were achieved compared to our method without super sampling.

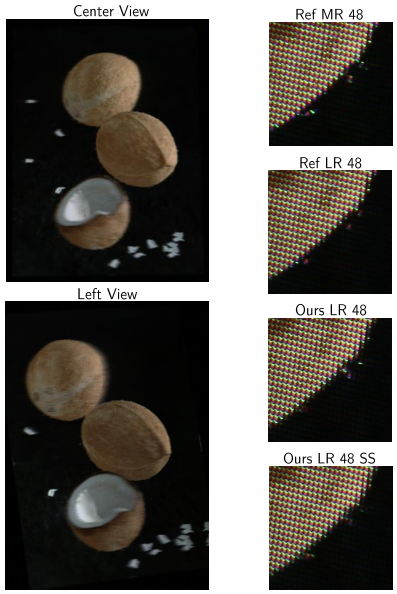


Fig. 13. Photographs of the Light Field Display

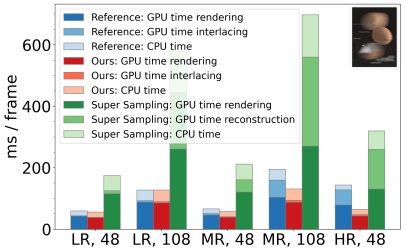


Fig. 14. Super sampling results. Time measurements.

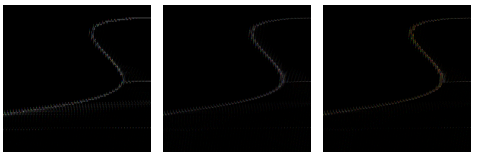


Fig. 15. Super sampling results. Close ups of the difference images examining the Teapot scene. The same close ups as in Figure 12a were used. (a) 10×|Ref HR - Ref LR|. (b) 10×|Ref HR - Ours HR SS|. (c) 10×|Ref HR - Ours HR|.

7 SUMMARY AND FUTURE WORK

In summary, we provide an in-depth analysis of the sampling behaviour during the interlacing step for parallax-based horizontal light field displays. Based on the findings of this analysis, we introduce a novel adaptation to the incorporated projective mappings that are tailored to the found sampling behavior. Thus, pixels never displayed are skipped efficiently. Lastly, we present a super sampling algorithm that further extends our method and considers display specifics even up to sub-pixel arrangement. Thus, quality improvements do not have to be coupled to a quadratic scale of memory consumption and render time, and instead can be controlled more fine-grained. As we only touch the virtual camera system, our method is compatible with most other optimizations for efficient light field display rendering.

In our evaluation in terms of quality, we limited our metrics to purely pixels-wise comparisons of the interlaced image. Future developments would profit from more elaborate evaluation techniques that also take into account the perceived image emitted by the display.

We see further refinements of our super sampling approach as a very promising direction. The mathematical framework would intuitively allow for a translation to an efficient temporal-anti-aliasing-like technique. For example, by incorporating a pseudo-random jitter in the horizontal and vertical skew parameters. Additionally, we found that a higher number of views leads to a sparser sampling along the sampling strips of the interlacing. This is a fact that is not exploited by our adaptations yet and is an interesting direction for future work.

Overall, our approach should be extensible to other parallax-based displays, like lenslet-based ones. This is because the sampling during rasterization can be adapted according to the underlying optical mechanism of these displays. Other display types, like tensor displays, cannot directly profit from the presented method.

Finally, we hope the presented algorithm helps to make rendering for light field displays in real-time more accessible. Thus, we aim to push this technology forward by allowing for even higher resolutions and views.

REFERENCES

Basit Ali et al. 2022. Microsoft ClearType - Typography. https://learn.microsoft.com/en-us/typography/cleartype/.

Mojtaba Bemana, Karol Myszkowski, Hans-Peter Seidel, and Tobias Ritschel. 2020. X-Fields: Implicit Neural View-, Light- and Time-Image Interpolation. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2020)* 39, 6 (2020). https://doi.org/10.1145/3414685.3417827

James R Bergen and Edward H Adelson. 1991. The Plenoptic Function and the Elements of Early Vision. *Computational models of visual processing* 1 (1991), 8.

Fabio Bettio, Enrico Gobbetti, Fabio Marton, and Giovanni Pintore. 2008. Scalable Rendering of Massive Triangle Meshes on Light Field Displays. *Computers & Graphics* 32, 1 (Feb. 2008), 55–64. https://doi.org/10.1016/j.cag.2007.11.002

Laura Fink, Sing Chun Lee, Jie Ying Wu, Xingtong Liu, Tianyu Song, Yordanka Velikova, Marc Stamminger, Nassir Navab, and Mathias Unberath. 2019. Lumipath–towards real-time physically-based rendering on embedded devices. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part V 22*. Springer, 673–681.

John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. 2019. DeepView: View Synthesis With Learned Gradient Descent. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE, Long Beach, CA, USA, 2362–2371. https://doi.org/10.1109/CVPR.2019.00247

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. 1996. The Lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 43–54. https://doi.org/10/dkzg3q

Yanxin Guan, Xinzhu Sang, Shujun Xing, Yuanhang Li, Duo Chen, and Xunbo Yu. 2020. Parallel Multi-View Polygon Rasterization for 3D Light Field Display. *Optics Express* 28, 23 (2020), 34406–34421. https://doi.org/10.1364/OE.408857

Yanxin Guan, Xinzhu Sang, Shujun Xing, Yuanhang Li, and Binbin Yan. 2019. Real-Time Rendering Method of Depth-Image-Based Multiple Reference Views for Integral Imaging Display. *IEEE Access* 7 (2019), 170545–170552. https://doi.org/10.1109/ACCESS.2019.2956102

Xiao Guo, Xinzhu Sang, Binbin Yan, Huachun Wang, Xiaoqian Ye, Shuo Chen, Huaming Wan, Ningchi Li, Zhehao Zeng, Duo Chen, Peng Wang, and Shujun Xing. 2022. Real-Time Dense-View Imaging for Three-Dimensional Light-Field Display Based on Image Color Calibration and Self-Supervised View Synthesis. *Opt. Express* 30, 12 (June 2022), 22260. https://doi.org/10.1364/OE.461789

Marc Levoy. 2006. Light Fields and Computational Imaging. *Computer* 39, 8 (Aug. 2006), 46–55. https://doi.org/10/cwxc5v

Dongxiao Li, Dongning Zang, Xiaotian Qiao, Lianghao Wang, and Ming Zhang. 2015. 3D Synthesis and Crosstalk Reduction for Lenticular Autostereoscopic Displays. *J. Display Technol.* 11, 11 (Nov. 2015), 939–946. https://doi.org/10.1109/JDT.2015.2405065

Documentation Looking Glass. 2022a. Light Fields. https://docs.lookingglassfactory.com/keyconcepts/capturing-a-lightfield.

Documentation Looking Glass. 2022b. Looking Glass Portrait. https://docs.lookingglassfactory.com/getting-started/portrait.

Documentation Looking Glass. 2022c. Quilts. https://docs.lookingglassfactory.com/keyconcepts/quilts.

Qungang Ma, Liangcai Cao, Zehao He, and Shengdong Zhang. 2019. Progress of Three-Dimensional Light-Field Display. *Chinese Optics Letters* 17, 11 (2019), 111001. https://doi.org/10.3788/COL201917.111001

Leonard McMillan and Gary Bishop. 1995. Plenoptic Modeling: An Image-Based Rendering System. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '95*. ACM Press, Not Known, 39–46. https://doi.org/10/drgrkd

Michael Oechsle, Michael Niemeyer, Lars Mescheder, Thilo Strauss, and Andreas Geiger. 2020. Learning implicit surface light fields. In *2020 International Conference on 3D Vision (3DV)*. IEEE, 452–462.

Bo Pang, Xinzhu Sang, Shujun Xing, Xunbo Yu, Duo Chen, Binbin Yan, Kuiru Wang, Chongxiu Yu, Boyang Liu, Can Cui, Yanxin Guan, Weikang Xiang, and Lei Ge. 2017. High-Efficient Rendering of the Multi-View Image for the Three-Dimensional Display Based on the Backward Ray-Tracing Technique. *Optics Communications* 405 (Dec. 2017), 306–311. https://doi.org/10.1016/j.optcom.2017.08.013

Sheng Shen, Shujun Xing, Xinzhu Sang, Binbin Yan, and Yingying Chen. 2022. Virtual Stereo Content Rendering Technology Review for Light-Field Display. *Displays* (2022), 102320. https://doi.org/10.1016/j.displa.2022.102320

Allen Sherrod. 2008. *Game Graphics Programming*. Course Technology/Charles River Media/Cengage Learning, Boston, MA.

Keith Stuart. 2011. Nintendo 3DS Draws Gamers Looking for 'glasses-Free' 3D Console. *The Guardian* (March 2011), 1.

Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. 2022. Light Field Neural Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Severin Todt, Christof Rezk-Salama, Andreas Kolb, and K-D Kuhnert. 2008. GPU-Based Spherical Light Field Rendering with Per-Fragment Depth Correction. *Computer Graphics Forum* 27, 8 (Dec. 2008), 2081–2095. https://doi.org/10/bqr4d8

Gordon Wetzstein, Douglas Lanman, Wolfgang Heidrich, and Ramesh Raskar. 2011. Layered 3D: Tomographic Image Synthesis for Attenuation-based Light Field and High Dynamic Range Displays. (2011), 1–12.

Gordon Wetzstein, Douglas Lanman, M Hirsch, and Ramesh Raskar. 2012. Real-Time Image Generation for Compressive Light Field Displays. *Journal of Physics* (2012).

Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. 2021. Nex: Real-time view synthesis with neural basis expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8534–8543.