Supplementary Material: Near-Eye Light Field Displays

Douglas Lanman David Luebke
NVIDIA Research

This document contains additional results and analysis in support of the primary text. Appendix A illustrates the prototype construction. Appendices B and C present a "backward compatibility" method for stereoscopic content. Appendices D and E describe PSNR analysis of retinal defocus and microdisplay defect correction, respectively.

A Photographs of the Head-Mounted Display Prototype

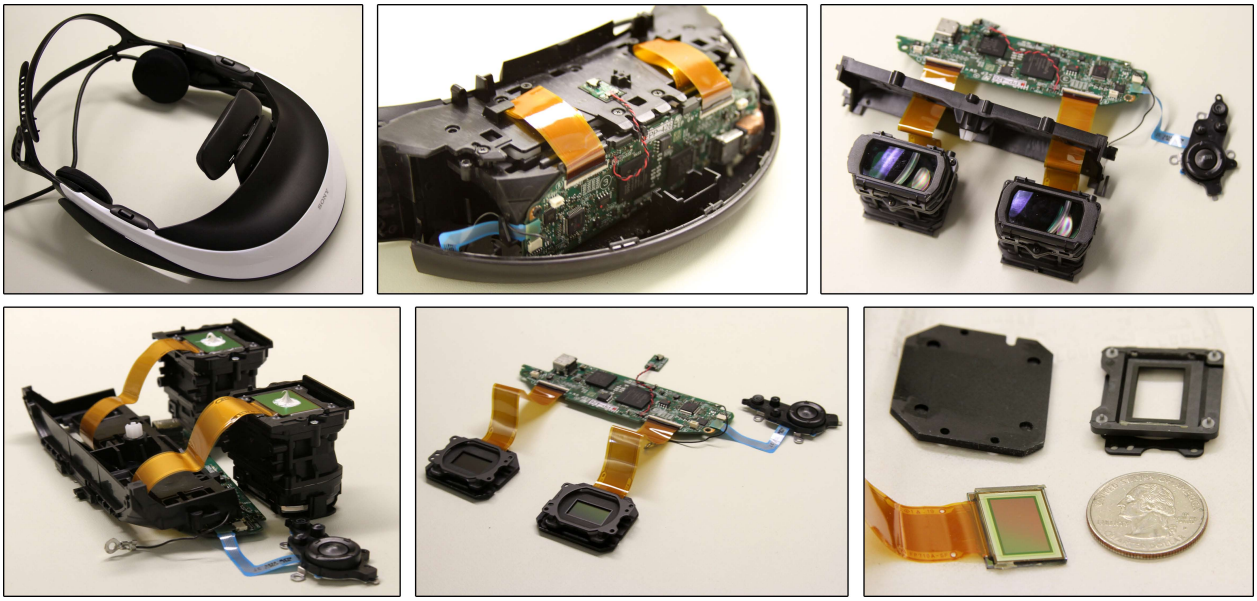

Figure S.1: *Disassembling the Sony HMZ-T1 personal media viewer. (Top Row) The case was removed, exposing the driver electronics and the eyepieces. (Bottom Row) The heat sink plates and magnifying optics were removed to access a pair of Sony ECX332A OLED microdisplays. On the right, a microdisplay is removed from its housing.*

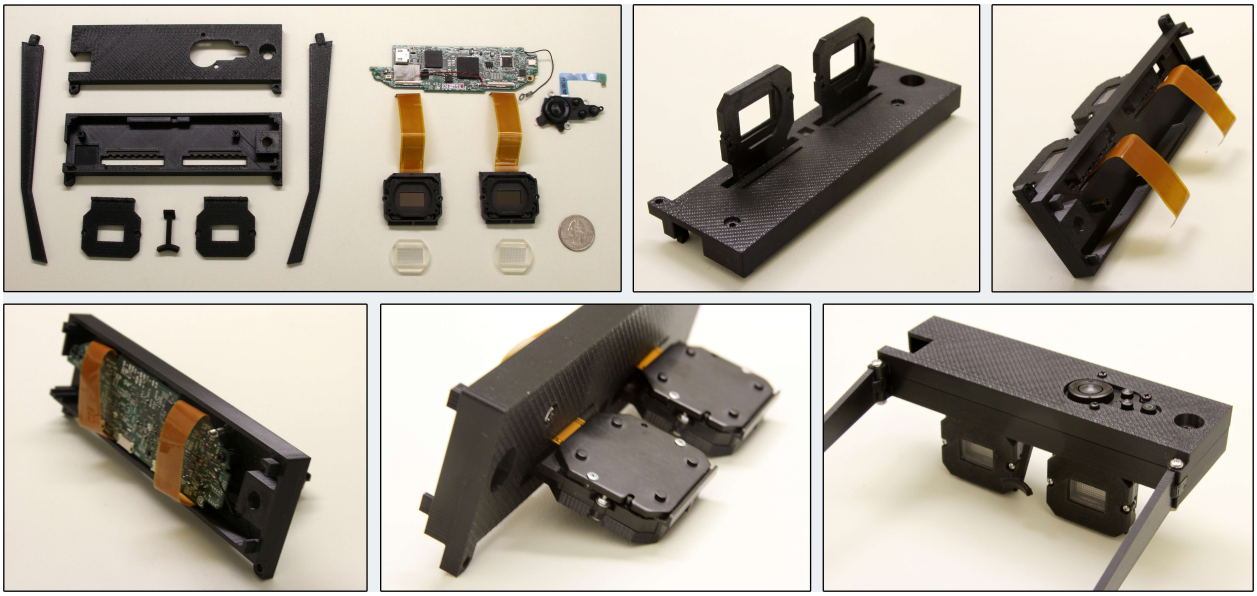

Figure S.2: *Assembling the near-eye light field HMD prototype. (Top Row) An enclosure was fabricated to hold the circuit board, pushbutton controller, and microdisplays. (Bottom Row) The OLED ribbon cables, only 8 cm in length, require the circuit board to be head-mounted; with longer cables, this box could be worn on the waist.*

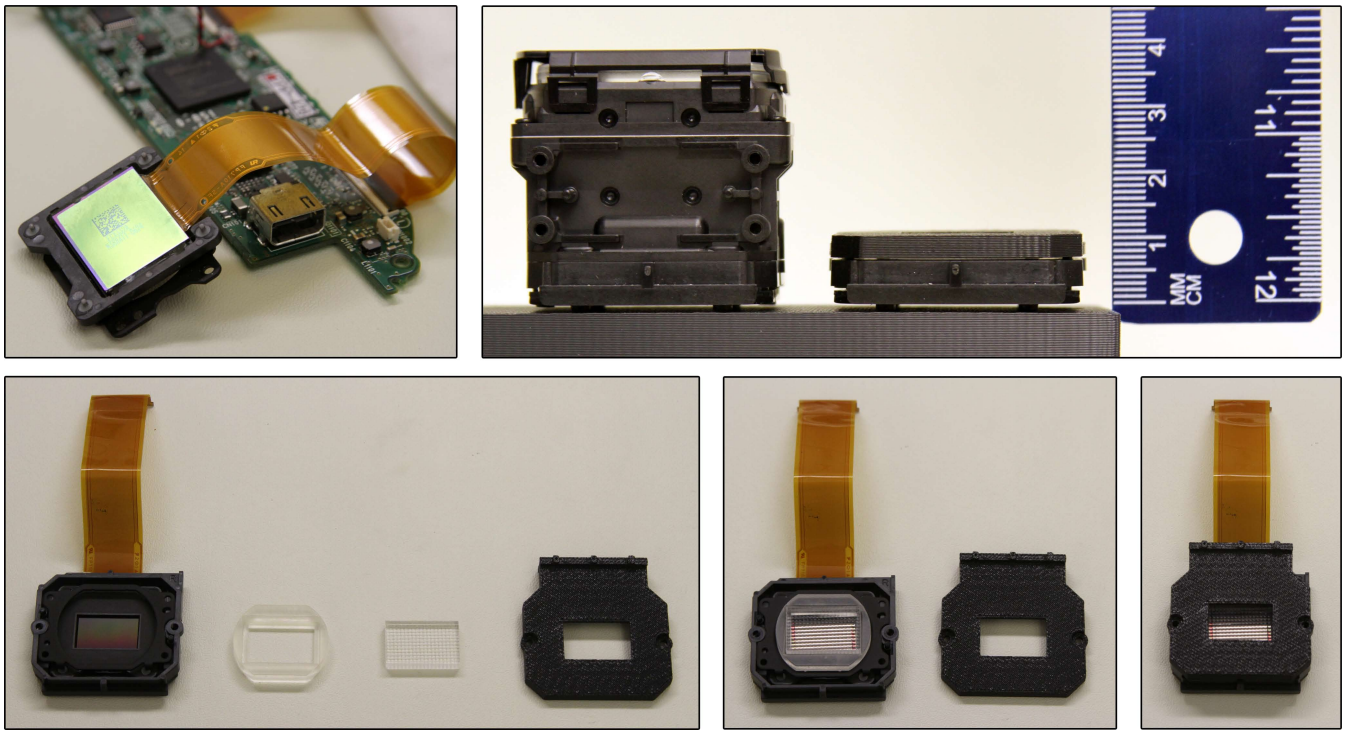

Figure S.3: *Assembling the near-eye light field display eyepiece. (Top Left) The microdisplay frame and heat sink plate (not shown) were retained in our eyepiece design. (Top Right) The original eyepiece and our modified microlens-based eyepiece are compared on the left and right, respectively. (Bottom Row) The near-eye light field display eyepiece contains four components, shown from left to right: an ECX332A OLED microdisplay, a custom-fabricated plastic stand-off, a Fresnel Technologies #630 microlens array, and a custom-fabricated plastic cover.*

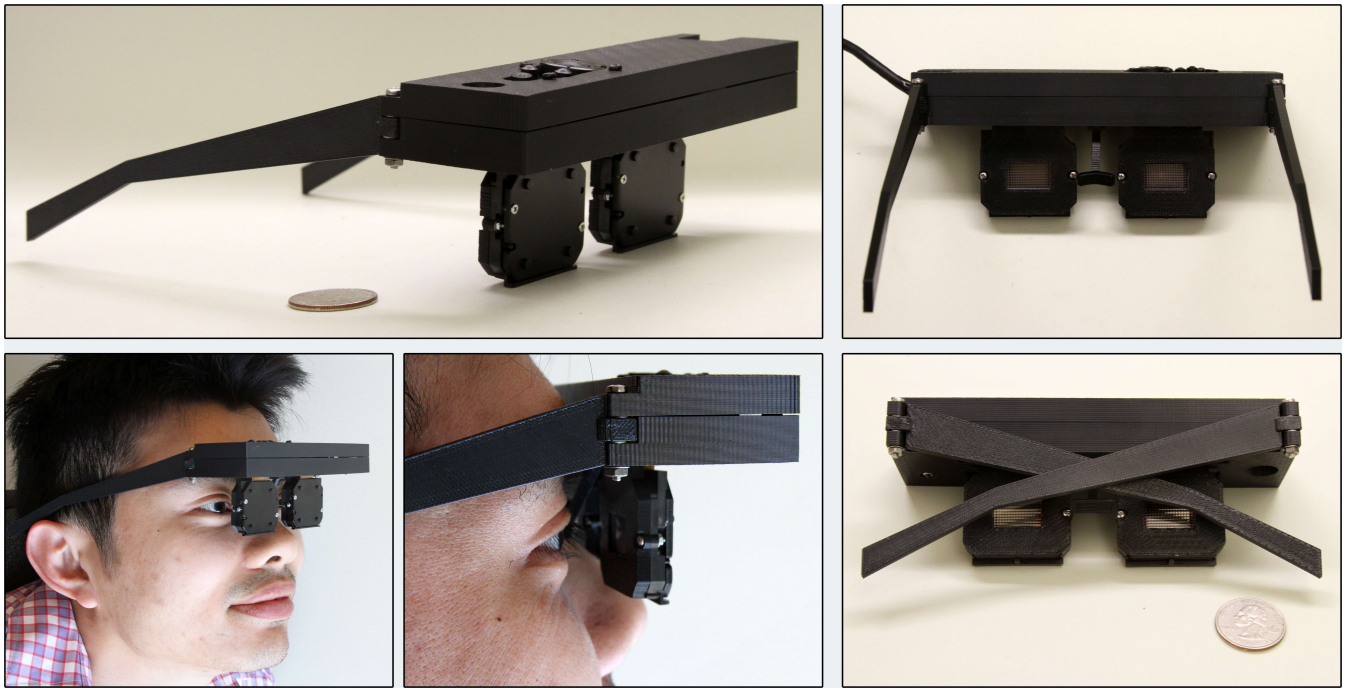

Figure S.4: *Our microlens-based design reduces form factor and weight. The prototype weighs 109 grams, in contrast to the 420-gram HMZ-T1. Components contribute as follows: enclosure (65.1 grams), circuit board and pushbutton controller (16.5 grams), and OLED with heat sink plate (11.8 grams). Each prototype eyepiece weighs 16.9 grams and is 1.0 cm thick, whereas the HMZ-T1 eyepiece weighs 69.4 grams and is 3.8 cm thick.*

B Backward Compatibility: Resampling Existing Stereoscopic Content

This appendix describes a method to present existing stereoscopic content using a binocular near-eye light field display, leveraging the programmable graphics pipeline. Section B.1 introduces a GLSL shader program optimized for resampling conventional 2D images for presentation by a microlens-based near-eye light field display. Section B.2 describes an associated anti-aliasing shader program for use with the proposed resampling scheme.

B.1 Resampling Shader Program

As shown in Figure S.6, rendering images for a binocular near-eye light field display requires assigning the color of each microdisplay pixel such that the emitted light field approximates the virtual scene. Following Section 4.2 in the primary text, ray tracing provides a direct solution. Consider the pixel located a distance x_s from the center of the left microdisplay. Assume this pixel is visible, to the viewer's left eye, through a microlens whose optical center is laterally displaced by a distance x_l from the center of the microdisplay. In this circumstance, the 3D coordinates of the microlens optical center and the microdisplay pixel define an optical ray, denoted by the red line in Figure S.6. Ray tracing provides a direct estimate of the microdisplay pixel color: casting the set of rays defined by every pixel located within the elemental image region corresponding to each microlens (see Section 3.2 in the primary text for the definition of elemental image regions). As shown in the supplementary video, we implement this rendering scheme using the NVIDIA OptiX GPU-accelerated ray tracing engine [Parker et al. 2010].

While ray tracing presents a flexible rendering alternative, a "backward compatibility" option is required to support existing stereoscopic sources, including movies and video games. Such sources have become widely available with the recent broad marketing of 3D television (3DTV). Similar to glass-free 3DTVs, particularly those using parallax barriers [Ives 1903] or lenticular arrays [Lippmann 1908], multiple views must be generated from stereoscopic sources. Proposed solutions include multiview synthesis, typically requiring estimating depth maps [Jin and Jeong 2008; Jain et al. 2011; Ramachandran and Rupp 2012]. However, unlike existing 3DTVs, near-eye light field displays necessitate rendering hundreds to thousands of views: one for each microlens. In the remainder of this section, we describe an alternate approach that is tailored to the distinct properties of near-eye light field displays.

We propose a simple backward compatibility option: **a binocular near-eye light field display emulates the appearance of a conventional stereoscopic display located a distance d_v from the viewer**. A viewer looking through a given eyepiece will perceive a rectangular virtual display surface, texture mapped with the corresponding view. Rather than ray trace a general 3D scene, conventional rasterization is applied to generate a stereoscopic pair: evaluating two off-axis perspective projections defined by the position and extent of the microlens arrays and the virtual stereoscopic display [Bourke 1999]. (Note that the left-eye frustum and right-eye frustum are shaded in blue and red in Figure S.6, respectively.) Under this model, the color of each microdisplay pixel is again determined by ray casting: interpolating the color at the point of intersection x_v with the virtual stereoscopic display surface. We recognize that this ray casting operation can be efficiently evaluated using a fragment shader program.

Listings S.1 and S.2 define the vertex and fragment shader programs (`fixed.vert` and `resample.frag`), written in GLSL, for an OpenGL-based implementation of our backward compatibility method. We assume that the application uses a pair of frame buffer objects (FBOs) to implement off-screen rendering, attaching the stereoscopic pair as textures to the left-eye and right-eye FBOs. During the final rendering pass, a single texture-mapped quadrilateral is drawn, embodying the virtual stereoscopic display surface. For quad-buffered rendering, the left-eye FBO texture is bound when drawing to the back left color buffer and vice versa for the back right color buffer. Note that Listing S.1 implements a minimal vertex shader that emulates the fixed-function graphics pipeline.

We briefly comment on the design of the `resample.frag` fragment shader defined in Listing S.2. The definition of the system architecture is provided via uniform variables on Lines 11–30. Lines 34–39 evaluate the displacement of the current fragment from the center of the microdisplay (i.e., distance x_s in Figure S.6). Similarly, Lines 41–45 evaluate the position of the optical center of the corresponding microlens (i.e., distance x_l in Figure S.6). Line 48

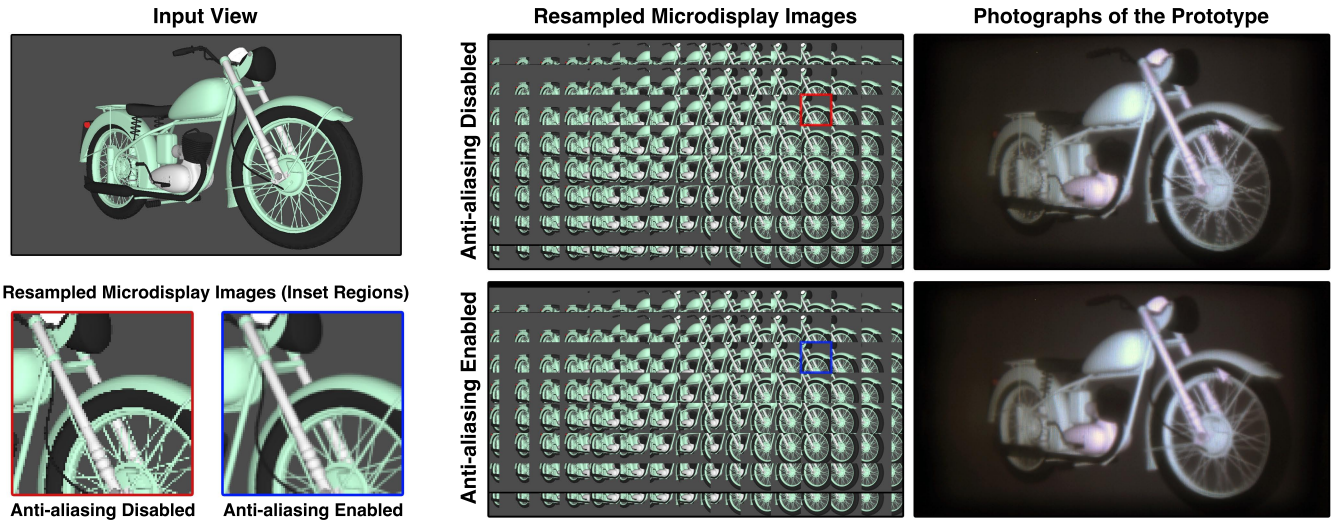

Figure S.5: *Resampling stereoscopic content. The backward-compatible shader program was used to resample stereoscopic image pairs for presentation using the binocular near-eye light field display prototype, described in Section 4.1 of the primary text. (Top Left) A right-eye view rendered using an OpenGL-based model viewer application. (Middle) Resampled microdisplay images generated using the resampling fragment shader program defined in Listing S.2. (Right) Corresponding photographs of the prototype near-eye light field display. (Bottom Left) Inset regions demonstrate the benefit of the anti-aliasing fragment shader defined in Listing S.3.*

evaluates the intersection of the corresponding optical ray with the virtual stereoscopic display surface; following Figure S.6, the horizontal coordinate of intersection is given by

$$x_v = x_l + \left(\frac{d_v - d_e}{d_l}\right)(x_s - x_l) - o_v \,. \tag{S.1}$$

Lines 50–61 evaluate the fragment color. Pixels outside the virtual stereoscopic display are set to black. As described in Section 3.2 of the primary text, artifacts result when rendering a portion of the virtual display visible through peripheral microlenses. The `microlensExcludeBorder` variable controls whether peripheral microdisplay regions are also set to black to prevent artifacts. Rendering results for a sample OpenGL application are shown in Figure S.5.

Our shader-based resampling method trades accommodation-convergence conflict resolution and defocus depth cues for backward compatibility. However, many of the benefits of general ray tracing persist. The distance d_v to the virtual display can be adjusted independently for each eye; similar to Pamplona et al. [2012], such freedom allows for the correction of the viewer's optical aberrations (i.e., their eyeglasses prescription) by locating the virtual display within the unaided accommodation range. Similarly, as discussed in Section 5.2 of the primary text, higher-order microlens aberrations can be corrected by modifying the ray casting operations implemented by the resampling shader program, allowing the microlens optical model to differ from that of a Gaussian thin lens approximation. However, we emphasize that efficient light field rendering methods remain a promising avenue of future research; either through "brute force" ray tracing or optimized multiview rendering schemes, such methods will be necessary to restore the central benefit of addressing accommodation-convergence conflict.

B.2 Anti-aliasing Shader Program

As described in Section 5.3 of the primary text, efficient light field anti-aliasing algorithms are also required for a practical near-eye light field display system. We implement an anti-aliasing pre-filter similar to that originally proposed by Levoy and Hanrahan [1996]. The left-eye and right-eye views are low-pass filtered, by (separable) convolution with a 2D box filter, before application of the resampling shader program. This operation is implemented using the two-pass shader program defined by the vertex and fragment shaders in Listings S.1 and S.3.

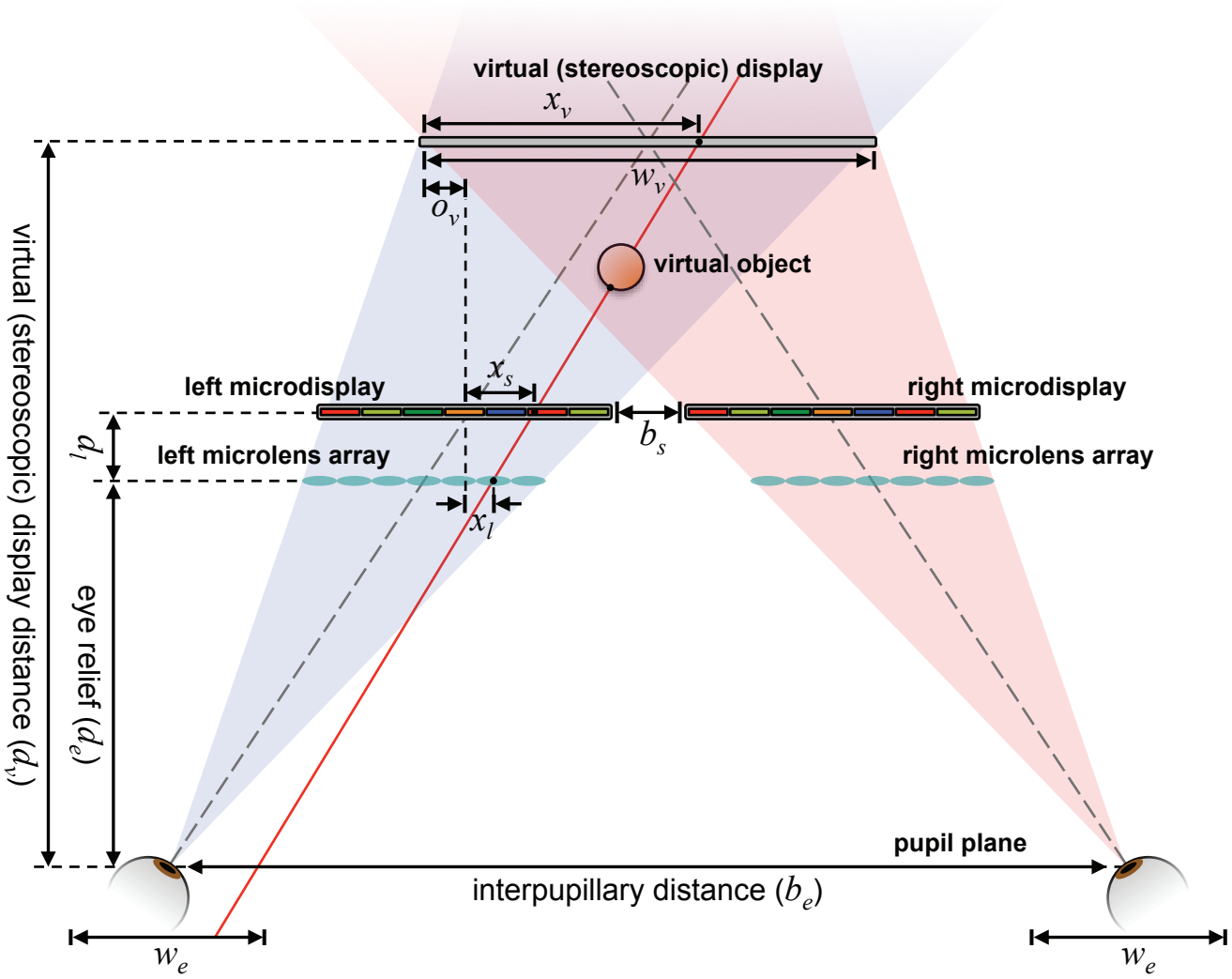

Figure S.6: *Rendering geometry for near-eye light field displays. A binocular near-eye light field display comprises a pair of microdisplays, each covered with a microlens array, separated by a distance b_s. The viewer's eyes are displaced by a distance d_e from the microlens arrays and separated by an interpupillary distance b_e. Rendering content for such displays requires assigning the color of each microdisplay pixel, either by direct ray tracing along the corresponding optical ray or by approximation using a virtual (stereoscopic) display (see Section B.1).*

```
/*
 *  fixed.vert
 *  Near-Eye Light Field Displays: Minimal "Fixed Function" Vertex Shader
 */
#version 120

void main(){

   // Evaluate interpolated texture coordinates.
   gl_TexCoord[0] = gl_MultiTexCoord0;

   // Emulate vertex transformation from fixed-function pipeline.
   gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;

}
```

Listing S.1: *A minimal vertex shader emulating the fixed-function graphics pipeline.*

```
/*
 *  resample.frag
 *  Near-Eye Light Field Displays: Resampling Fragment Shader
 */
#version 120

// General variables
uniform sampler2D viewTexture;              // sampler corresponding to input view

// Microdisplay variables
uniform vec2      microdisplayDim;          // physical dimensions (mm) [w_s]
uniform int       microdisplayOrientation;  // orientation (0: upright, 1: inverted)

// Microlens array variables
uniform float     microlensPitch;           // pitch (mm) [w_l]
uniform float     microlensDist;            // distance from microdisplay (mm) [d_l]
uniform float     microlensRotation;        // rotation angle (radians)
uniform vec2      microlensOffset;          // alignment offsets (mm)
uniform bool      microlensExcludeBorder;   // exclude partial microlenses along microdisplay border

// Elemental image variables
uniform float     elementalImagePitch;      // pitch (mm) [Δw_s]
uniform vec2      elementalImageOffset;     // offset from microdisplay origin (mm)
uniform vec4      elementalImageRange;      // index range (min column, max column, min row, max row)

// Virtual screen variables
uniform vec2      screenDim;                // physical dimensions (mm) [w_v]
uniform float     screenDist;               // distance from microlens array (mm) [d_v]
uniform vec2      screenPosition;           // offset from microdisplay origin (mm) [o_v]

void main(){

   // Evaluate microdisplay sample coordinates [x_s].
   vec2 microdisplayCoord;
   if(microdisplayOrientation == 0)
      microdisplayCoord = microdisplayDim*(gl_TexCoord[0].st-0.5);
   else
      microdisplayCoord = microdisplayDim*(0.5-gl_TexCoord[0].st);

    // Evaluate corresponding microlens coordinates (i.e., microlens pixel is observed through) [x_l].
   mat2 R = mat2( cos(microlensRotation), -sin(microlensRotation),
                  sin(microlensRotation),  cos(microlensRotation) );
   vec2 microlensIndex = floor(R*(microdisplayCoord-elementalImageOffset)/elementalImagePitch);
   vec2 microlensCoord = transpose(R)*microlensPitch*microlensIndex + microlensOffset;

   // Evaluate virtual screen sample coordinates [x_v].
   vec2 screenCoord = microlensCoord + (screenDist/microlensDist)*(microdisplayCoord-microlensCoord) - ↩
       screenPosition;

   // Assign color of resampled view.
   vec2 texCoord = screenCoord/screenDim;
   if( ((texCoord.x>0.0) && (texCoord.x<1.0)) &&
       ((texCoord.y>0.0) && (texCoord.y<1.0)) ){
      if( !microlensExcludeBorder ||
          ( ((microlensIndex.x>elementalImageRange.x) && (microlensIndex.x<elementalImageRange.y)) &&
            ((microlensIndex.y>elementalImageRange.z) && (microlensIndex.y<elementalImageRange.w)) ) )
         gl_FragColor = vec4(texture2D(viewTexture, texCoord).rgb, 1);
      else
         gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
    } else
       gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);

}
```

Listing S.2: *A fragment shader for resampling existing stereoscopic content for near-eye light field displays. Note that commented variables enclosed by square brackets denote the correspondence of shader variables with the labeled dimensions in Figure S.6 and Figure 4 in the primary text.*

```
1  /*
2   *  antialias.frag
3   *  Near-Eye Light Field Displays: Anti-aliasing Fragment Shader
4   */
5  #version 120
6
7  // General variables
8  uniform sampler2D viewTexture;         // sampler corresponding to input view
9
10 // Anti-aliasing variables
11 uniform int       antialiasDirection;  // filter direction (0: horizontal, 1: vertical)
12 uniform float     antialiasLength;     // filter length in normalized texture coordinate units
13 uniform int       antialiasNumSamples; // number of samples along filter length
14
15 void main(){
16
17    // Evaluate texture coordinate.
18    vec2 p0 = gl_TexCoord[0].st;
19
20    // Apply separable anti-aliasing filter, if necessary.
21    if(antialiasNumSamples > 0){
22
23       // Evaluate temporary anti-aliasing variables.
24       float antialiasSampleOffset = antialiasLength/float(antialiasNumSamples-1);
25       float antialiasHalfLength = antialiasLength/2.0;
26       float antialiasWeight = 1.0/float(antialiasNumSamples);
27       vec3 color = vec3(0.0, 0.0, 0.0);
28       vec2 p;
29
30       // Apply 1D filter along horizontal or vertical axis.
31       if(antialiasDirection == 0){
32          for(int i=0; i<antialiasNumSamples; i++){
33             p = p0 + vec2(antialiasSampleOffset*float(i)-antialiasHalfLength, 0.0);
34             color += antialiasWeight*texture2D(viewTexture, p).rgb;
35          }
36       } else{
37          for(int i=0; i<antialiasNumSamples; i++){
38             p = p0 + vec2(0.0, antialiasSampleOffset*float(i)-antialiasHalfLength);
39             color += antialiasWeight*texture2D(viewTexture, p).rgb;
40          }
41       }
42       gl_FragColor = vec4(color, 1.0);
43
44    } else{
45
46       // No anti-aliasing is required, so pass through input color.
47       gl_FragColor = vec4(texture2D(viewTexture, p0).rgb, 1.0);
48    }
49 }
```

Listing S.3: *A fragment shader for anti-aliasing images for near-eye light field displays. Full-scene anti-aliasing (FSAA) is implemented as a separable convolution with a box filter. The filter length is determined by the size of a microdisplay pixel, as projected into the virtual (stereoscopic) display plane shown in Figure S.6 (see Equation S.2).*

Listing S.3 implements full-scene anti-aliasing (FSAA). Stereoscopic views are assumed to be oversampled, relative to the resolution of the near-eye light field display (given by Equation 8 in the primary text). The low-pass filter length $\hat{p}_v$ is determined by the microdisplay pixel size, as magnified by projection into the virtual display plane; defined as the `antialiasLength` uniform variable on Line 13 of Listing S.3, this parameter is given by

$$\hat{p}_v = \frac{Mp}{w_v} = \left(\frac{d_v - d_e}{d_l}\right)\left(\frac{p}{w_v}\right), \tag{S.2}$$

where the length is expressed in normalized texture coordinate units and M corresponds to the magnification factor given by Equation 2 in the primary text. Anti-aliasing results are shown in Figure S.5.

C Case Study: Modifying *Doom 3* to Support Near-Eye Light Field Displays

This appendix describes how *Doom 3 BFG Edition* was modified to support near-eye light field displays using the backward-compatible shader programs in Appendix B. The source code was recently released and can be downloaded from the official GitHub repository[1]. As stated in the release notes, the source archive does not include game data, which may be purchased separately[2]. We emphasize that this video game supports quad-buffered stereoscopic rendering, providing an ideal platform to demonstrate our shader-based resampling method.

Listing S.4 includes the necessary fragment shader, adapted from the `base/renderprogs/stereoWarp.pixel` shader provided to support the Oculus Rift. (In the following discussion, file paths are defined relative to the root source directory.) Note that this shader combines the resampling and anti-aliasing functions into a single-pass implementation, minimizing the required alterations to the source code. To integrate this shader, we chose to add a second quad buffering mode to `neo/renderer/OpenGL/gl_backend.cpp`, binding our custom fragment shader before drawing to the back left and back right color buffers. The near-eye light field display architecture was defined using the `RENDERPARM_USER` register indices enumerated in `neo/renderer/RenderProgs.h`. As a result, Lines 7–38 interpret these user variables to assign the display parameters. For brevity, the `microdisplayOrientation` and `microlensExcludeBorder` variables, previously appearing in Listing S.2, have been omitted from this implementation. Lines 40–65 define the joint resampling and anti-aliasing operations, formatted to comply with the *Doom 3* fragment shader conventions.

We emphasize that Listing S.4 implements single-pass anti-aliasing, rather than the more computationally-efficient two-pass method described in Listing S.3. Specifically, a non-separable 2D convolution is performed. As a result, the necessary source revisions are primarily localized to `gl_backend.cpp`; however, as tabulated in Figure S.7, this simplicity comes at the cost of reduced frame rates (orange line). We observe resampling, without anti-aliasing, has a minimal impact on frame rates (red line). To estimate the benefits of the proposed two-pass anti-aliasing filter, we modified Listing S.4 to only filter along the horizontal axis (green line).

In summary, our shader programs provide a practical route to upgrading existing stereoscopic sources to support near-eye light field displays. See Figure S.8 and the video accompanying this appendix to review photographs and videos recorded during a brief gaming session.

[1] `https://github.com/id-Software/DOOM-3-BFG/`
[2] `http://store.steampowered.com/app/208200/`

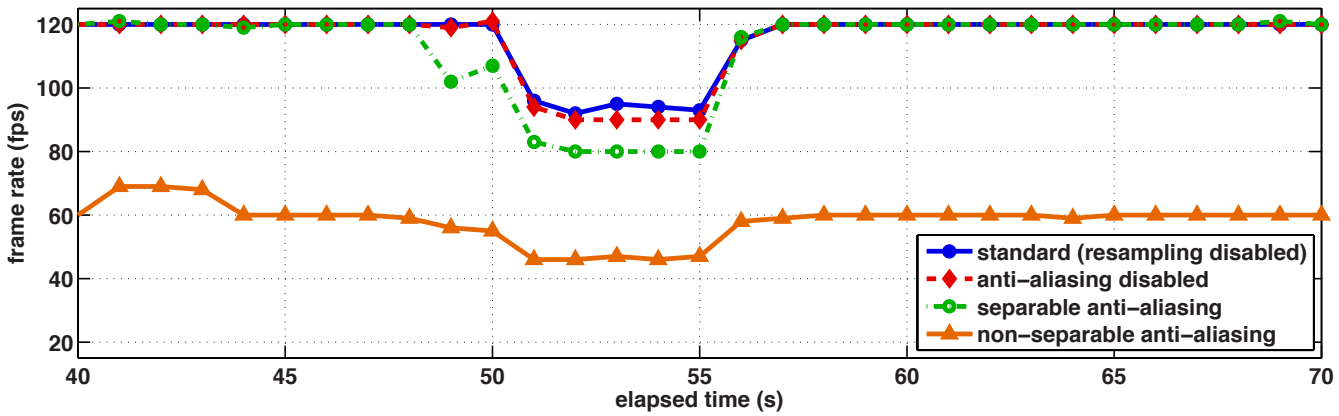

Figure S.7: *Frame rates for Doom 3 BFG Edition using the combined resampling and anti-aliasing shader program in Listing S.4. Frame rates are limited to a maximum of 120 fps by the application. The resampling shader, modified from Listing S.2, has a negligible impact on performance when anti-aliasing is disabled or when implemented via a separable convolution. As implemented in Listing S.4, anti-aliasing using a non-separable convolution halves the frame rate to 60 fps. (Frame rates were measured using the system described in Section 4.2 of the primary text and were recorded during the introductory sequence shown in the supplementary video accompanying this case study.)*

```
/*
 * stereoResample.pixel
 * Doom 3 BFG Edition Fragment Shader for Near-Eye Light Field Displays
 */
#include "global.inc"

// Define uniform variables and input/output structures.
uniform sampler2D samp0 : register(s0);   uniform sampler2D samp1 : register(s1);
uniform float4  rpUser0 : register(c128); uniform float4  rpUser1 : register(c129);
uniform float4  rpUser2 : register(c130); uniform float4  rpUser3 : register(c131);
uniform float4  rpUser4 : register(c132); uniform float4  rpUser5 : register(c133);
uniform float4  rpUser6 : register(c134); uniform float4  rpUser7 : register(c135);
struct PS_IN  { float2 texcoord0 : TEXCOORD0_centroid; };
struct PS_OUT { float4 color : COLOR; };

void main(PS_IN fragment, out PS_OUT result ) {

   // Microdisplay variables
   const vec2  microdisplayDim      = rpUser0.xy; // physical dimensions (mm)

   // Microlens array variables
   const float microlensPitch       = rpUser0.w;  // pitch (mm)
   const float microlensDist        = rpUser1.x;  // distance from microdisplay (mm)
   const float microlensRotation    = rpUser1.y;  // rotation angle (radians)
   const vec2  microlensOffset      = rpUser1.zw; // alignment offsets (mm)

   // Elemental image variables
   const float elementalImagePitch  = rpUser2.y;  // pitch (mm)
   const vec2  elementalImageOffset = rpUser2.zw; // offset from microdisplay origin (mm)

   // Virtual screen variables
   const vec2  screenDim            = rpUser4.xy; // physical dimensions (mm)
   const vec2  screenPosition       = rpUser4.zw; // offset from microdisplay origin (mm)
   const float screenDist           = rpUser5.x;  // distance from microlens array (mm)

   // Anti-aliasing variables
   const vec2  antialiasLength      = rpUser6.xy; // filter lengths
   const vec2  antialiasNumSamples  = rpUser6.zw; // number of samples

   // Evaluate microdisplay, microlens, and virtual screen sample coordinates.
   vec2 microdisplayCoord = microdisplayDim*(fragment.texcoord0-0.5);
   mat2 R = mat2( cos(microlensRotation), -sin(microlensRotation),
                  sin(microlensRotation),  cos(microlensRotation));
   vec2 microlensIndex = floor(R*(microdisplayCoord-elementalImageOffset)/elementalImagePitch);
   vec2 microlensCoord = transpose(R)*microlensPitch*microlensIndex + microlensOffset;
   vec2 screenCoord = microlensCoord + (screenDist/microlensDist)*(microdisplayCoord-microlensCoord) - ↩
       screenPosition;

   // Evaluate temporary anti-aliasing variables.
   vec2 antialiasSampleOffset = antialiasLength/float(antialiasNumSamples-1);
   vec2 antialiasHalfLength   = antialiasLength/2.0;
   float antialiasWeight      = 1.0/(float(antialiasNumSamples.x)*float(antialiasNumSamples.y));

   // Assign color of resampled view with optional anti-aliasing by non-separable convolution.
   vec2 texCoord0 = screenCoord/screenDim; vec2 texCoord; float3 color = float3(0, 0, 0);
   if( ((texCoord0.x>0.0) && (texCoord0.x<1.0)) && ((texCoord0.y>0.0) && (texCoord0.y<1.0)) ){
      for(int i=0; i<antialiasNumSamples.x; i++){
         for(int j=0; j<antialiasNumSamples.y; j++){
            texCoord = texCoord0 + vec2(antialiasSampleOffset.x*float(i)-antialiasHalfLength.x,
                                        antialiasSampleOffset.y*float(j)-antialiasHalfLength.y);
            result.color += antialiasWeight*tex2D(samp0, texCoord);
         }
      }
   } else
      result.color = float4(0, 0, 0, 1);
}
```

Listing S.4: *A fragment shader to add support for near-eye light field displays to Doom 3 BFG Edition.*

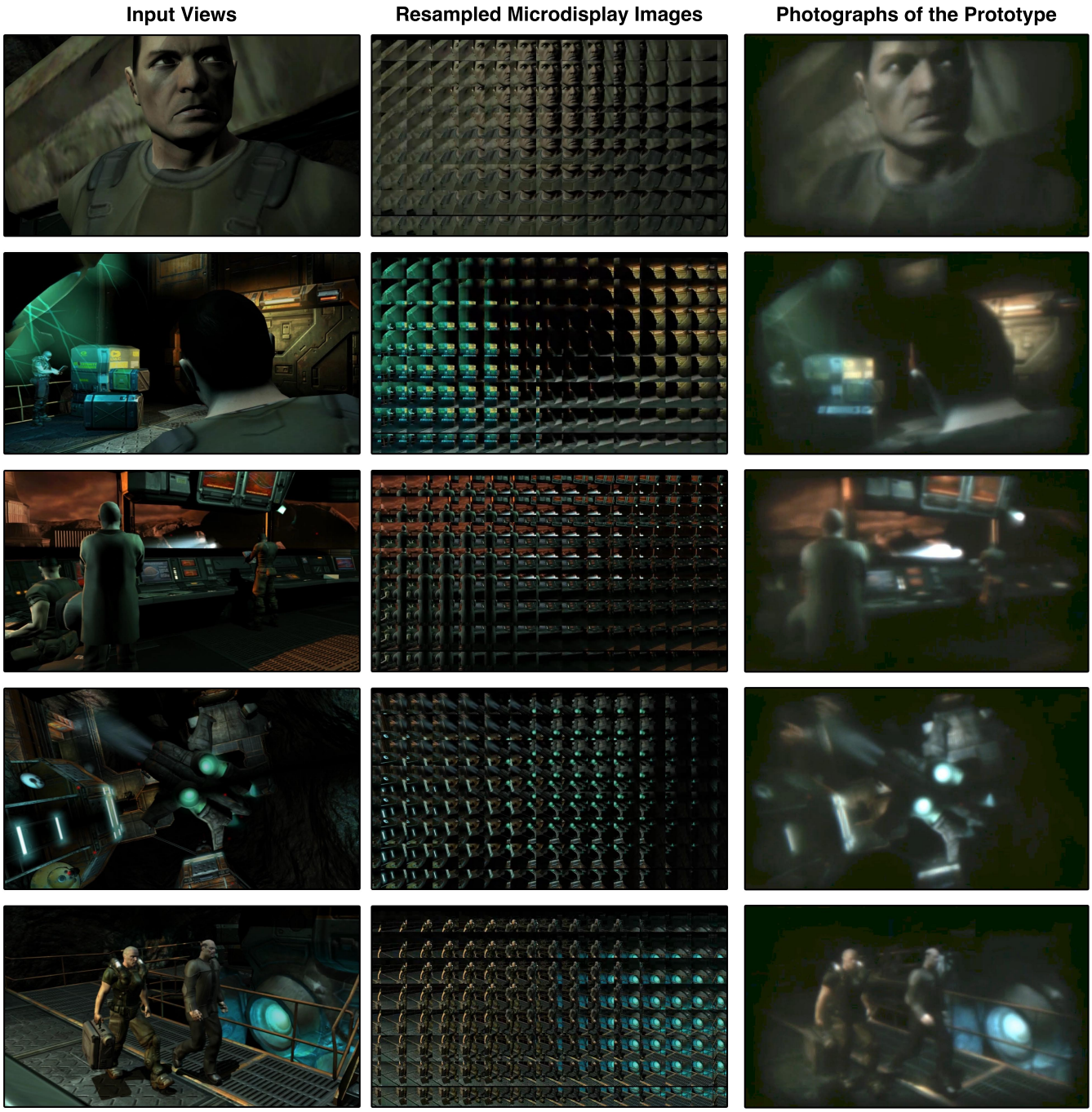

Figure S.8: *Modifying Doom 3 BFG Edition for the binocular near-eye light field display prototype. (Left) The input right-eye view, as rendered to support standard quad-buffered stereo displays (e.g., the Sony HMZ-T1 described in Section 4.1 of the primary text). (Middle) Resampled, anti-aliased microdisplay images generated using the custom fragment shader program in Listing S.4. (Right) Corresponding photographs of the prototype near-eye light field display. (Note that these examples are stills taken from the video accompanying this appendix.)*

D Extended PSNR Analysis of Retinal Defocus

This section elaborates on the peak signal-to-noise ratio (PSNR) analysis of retinal defocus blur presented in Section 5.1 of the primary text. In this section, we use a numerical simulation of near-eye light field displays to assess their ability to approximate retinal defocus blur, as resulting from the observation of out-of-focus scene elements. In the following analysis, ideal retinal images were simulated using Equation 14 in the primary text. A ray tracing model, implemented in Matlab, was used to simulate retinal images for the OLED-based prototype described in Appendix A and Section 5.1 of the primary text. The approximation error was averaged over the set of target images in Figure S.10, where each image was assigned as a texture to a virtual plane located a distance d_o from a viewer whose eye was accommodated at a distance d_a. Retinal defocus approximation errors are tabulated in Figure S.9.

Figures S.11 and S.12 provide visual examples for interpreting various points within Figure S.9. In these examples, the virtual object plane was fixed at a distance of 66.7 cm (1.50 diopters). When the viewer accommodates on the virtual object plane (i.e., the black line in Figure S.9), we observe that the approximation error is dominated by "pixelization" artifacts (resulting from the limited resolution of the prototype, relative to an ideal high-resolution display); as illustrated on the third, fourth, and fifth rows of Figures S.11 and S.12, when "close" to focusing on the virtual object, the near-eye light field display achieves a maximum resolution of 146×78 pixels, following Section 4.1 in the primary text. Thus, we observe pixelization artifacts predominate within the "zone of comfort" estimated by Shibata et al. [2011] (the boundary of which is denoted by dashed white lines in Figure S.9).

As described by Shibata et al., the zone of comfort corresponds to the range of comfortable combinations of vergence and focal distance. We observe that the PSNR of the retinal blur approximation rises markedly for combinations of virtual object distances (i.e., vergence distances) and accommodation distances outside this zone of comfort. As a result, near-eye light field displays enhance retinal blur depiction precisely within the zone where accommodation-convergence conflict occurs. We emphasize that perceptual trials are a promising direction to confirm whether such objective measures of improvement agree with subjective judgements of viewing comfort by human observers.

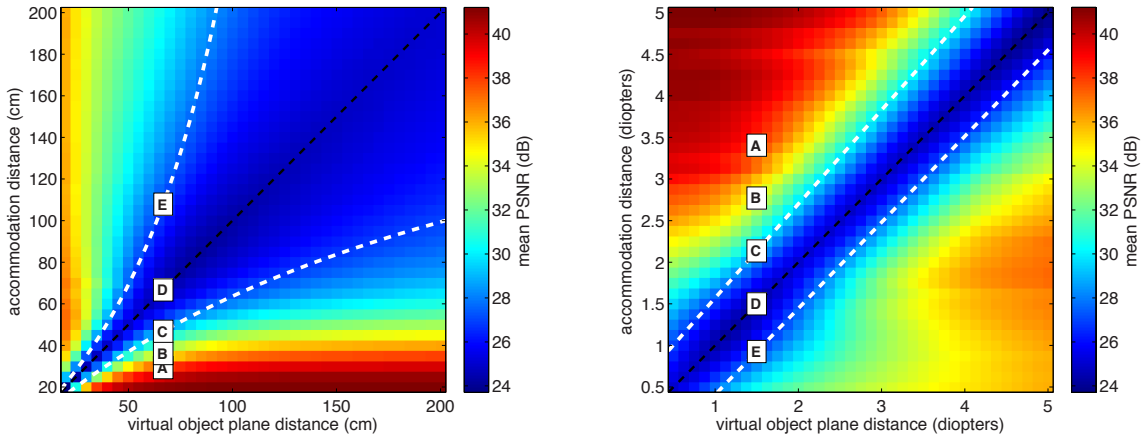

Figure S.9: *Approximating retinal defocus blur using near-eye light field displays. Approximation PSNR is tabulated as a function of virtual object distance and accommodation distance, expressed in centimeters (left) and diopters (right). The dashed black line denotes "natural viewing", for which accommodation distance equals virtual object distance. Dashed white lines denote the boundary of the "zone of comfort" estimated by Shibata et al. [2011]. Labeled points define the conditions for the corresponding simulated retinal images shown in Figures S.11 and S.12.*

Figure S.10: *Target image set used to evaluate the approximation of retinal defocus blur in Figure S.9.*

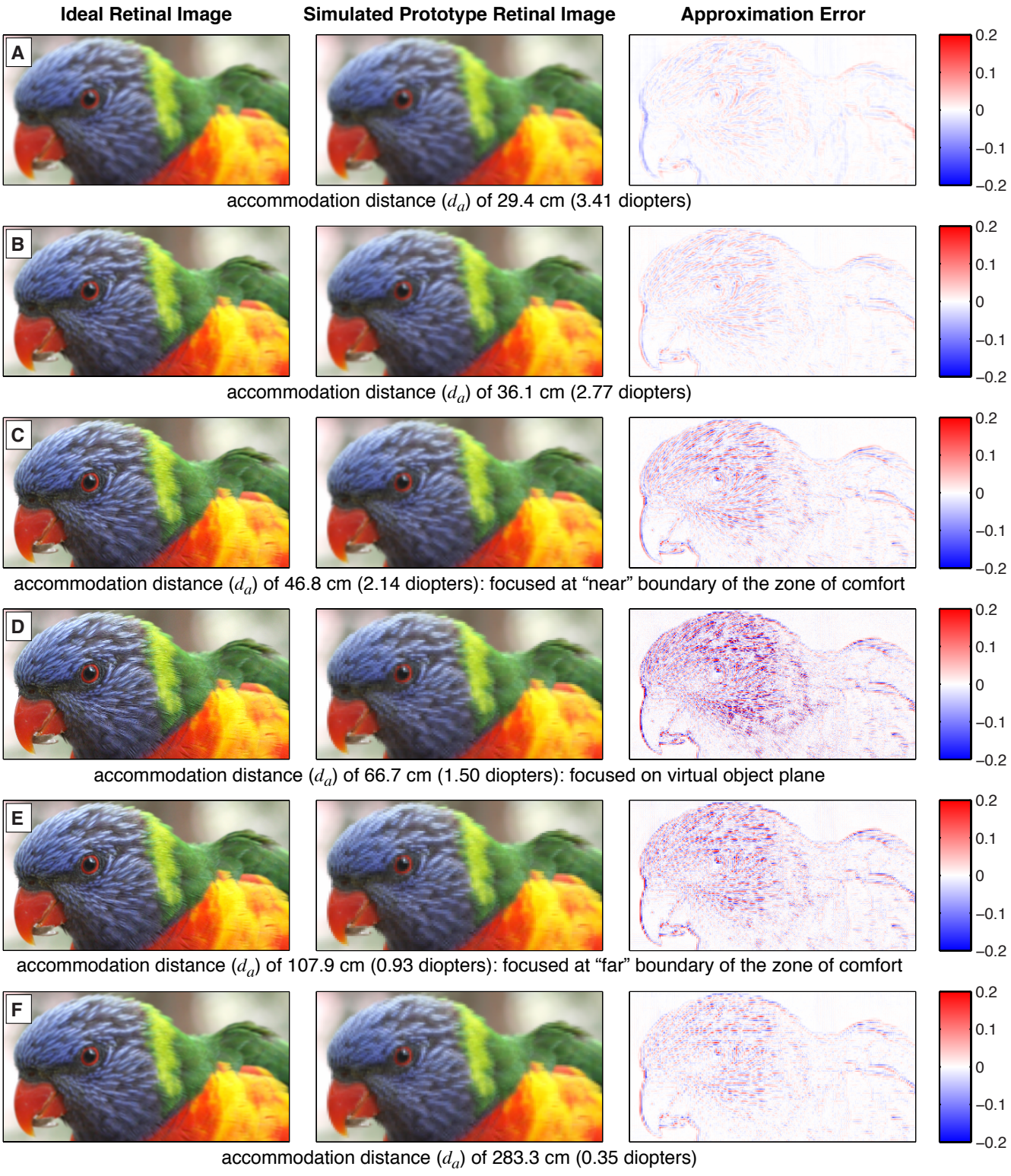

Figure S.11: *Simulating retinal defocus blur. Ideal retinal images were simulated using Equation 14 from the primary text. A ray tracing model, implemented in Matlab, approximated retinal images for the OLED-based prototype described in Appendix A and Section 4.3 of the primary text. Labels in the top-left corner of each row denote correspondences with points in Figure S.9. For this example, the virtual object plane is located at a distance of 66.7 cm (1.50 diopters). The third and fifth rows illustrate simulated retinal images for accommodation distances located at the boundaries of the “zone of comfort” estimated by [2011].*

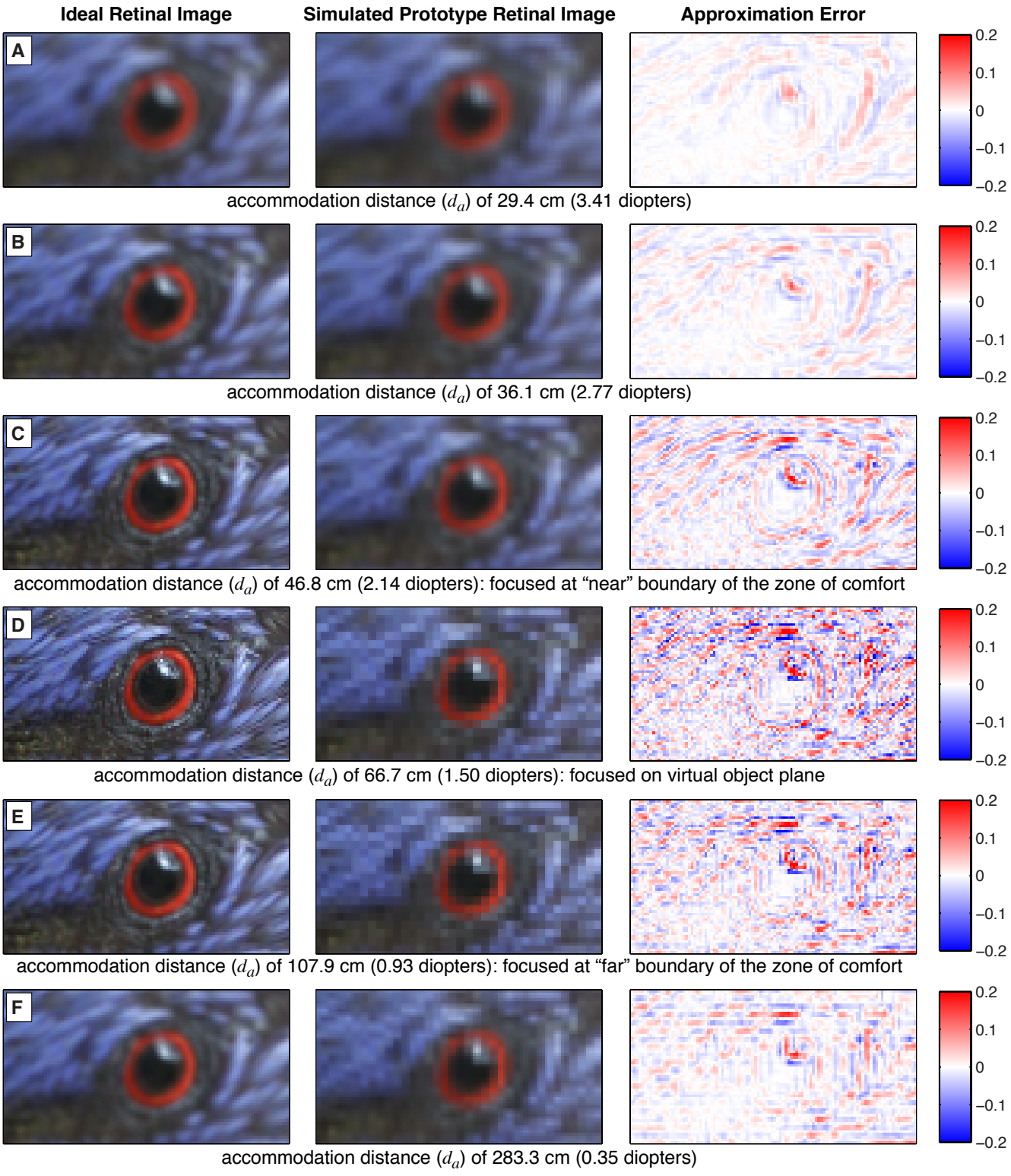

Figure S.12: *Magnified regions of the images in Figure S.11, centered on the eye of the bird. Retinal defocus blur is accurately approximated by the near-eye light field display, here utilizing a maximum of 4×4 overlapping views, following Sections 3.3 and 5.1 of the primary text. Note that, when the viewer is focused on the virtual object plane (i.e., the example in the fourth row), approximation errors are predominated by pixelization artifacts due to the limited spatial resolution of 146×78 pixels achieved by the near-eye light field display prototype.*

E Microdisplay Defect Correction

This appendix expands on the discussion of microdisplay defect correction presented in Section 5.2 of the primary text. To realize practical near-eye display applications, a viable path to large-format microdisplay fabrication is required. We propose a manufacturing approach similar to that adopted with medium-format digital image sensors: utilizing multiple photolithographic masks to overcome the current reticle area limitations. As with such medium-format sensors, we further recognize fabrication costs can best be controlled by developing display algorithms that are tolerant to defective pixels. Here we present one solution: **utilizing a computationally-efficient tomographic optimization framework to exploit redundancy within the elemental image array to compensate for defects**.

As listed in Table S.1, the ISO 13406-2 standard identifies three types of flat panel display defects [ISO 2000]. A type-1 defect corresponds to a "hot" pixel (i.e., a group of color subpixels that are continuously illuminated). A type-2 defect is a so-called "dead" pixel (i.e., appearing continuously dark). Finally, a type-3 defect is a single color subpixel that is either continuously on or off. ISO 13406-2 classifies flat panel displays by the maximum allowed density of such defects, with additional constraints on the allowed spacing of clusters of various defect types. Microdisplays with varying densities of simulated pixel defects are shown in Figures S.13 and S.14.

We recognize that near-eye light field displays present a unique opportunity to correct for defective pixels—a property not shared with conventional HMD designs employing simple magnifiers. Each scene point appears in multiple elemental images; for example, in the first row of Figure S.8, the tip of the man's nose is reproduced in 24 elemental images. Each of these microdisplay pixels produces a distinct ray of the emitted light field. This bundle of rays, which passes through differing microlenses but within the viewing eye box, converges close to the position of the virtual scene point. For a diffuse point, the radiances should be nearly identical; even for specular points, variation will be moderate (due to the small solid angle defined by the viewing eye box and the virtual point). As a result, redundancy in the elemental image array provides a simple mechanism to compensate for defects: distributing the error due to any given defect among the microdisplay pixels corresponding with the same scene point.

To demonstrate the potential of microdisplay defect correction, we present an illustrative example: correcting defects for planar scenes, such as those depicted using the "backward compatible" shader program presented in Appendix B. In the following discussion, the microdisplay pixels are represented as a column vector $\mathbf{m}$ of length N, with each element m_n representing the radiant emittance of a pixel. Similarly, the ideal retinal image is modeled as a column vector $\mathbf{r}$ of length M. We assume, under ideal circumstances, that the position and size of the viewer's pupil is known. As a result, the irradiance r_m at a given point in the retinal image is modeled as a weighted summation of microdisplay pixel emittances (corresponding with the integration of the set of optical rays entering the pupil and impinging on the retinal image point). This image formation model is formally expressed as:

$$r_m = \sum_{n=1}^{N} P_{mn}\, m_n, \tag{S.3}$$

where $\mathbf{P}$ is an $M \times N$ projection matrix defining the contributions of microdisplay pixels to each retinal image point.

A defect correction algorithm provides a *defect-corrected microdisplay image* $\hat{\mathbf{m}}$ such that the retinal image is "optimally" reproduced. In practice, optimality should be defined using a perceptual error metric. However, in this

Defect Type	Class I	Class II	Class III	Class IV	Figure S.13	Figure S.14
Type 1 ("hot" pixel)	0	2	5	50	500	5,000
Type 2 ("dead" pixel)	0	2	15	150	1,500	15,000
Type 3 ("stuck" subpixel)	0	5	50	500	5,000	50,000

Table S.1: *The ISO 13406-2 standard classifies displays by the maximum number of various defects (per 1 million pixels). Defect densities in Figures S.13 and S.14 exceed "class IV" panels by factors of 10 and 100, respectively.*

s

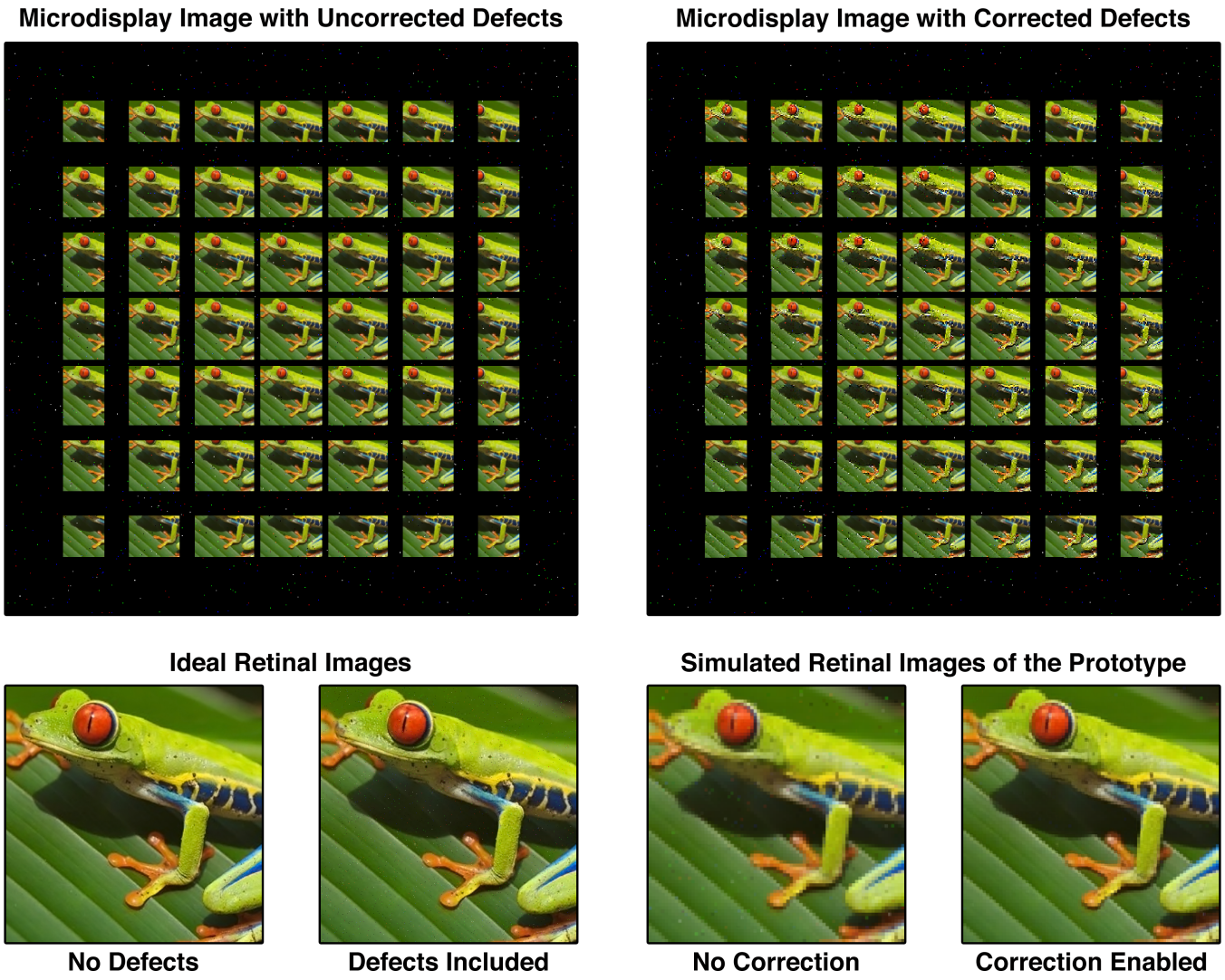

Figure S.13: *Correcting microdisplay defects. Defect densities are 10 times greater than allowed for a "class IV" display, as defined in Table S.1. (Top) Simulated microdisplay images, corresponding with the HMD prototype. Zoom in to inspect alterations made by the defect correction algorithm. Note that corrections occur within circular regions, corresponding to the projections of the viewer's pupil. (Bottom) Ideal and simulated prototype retinal images.*

appendix, we demonstrate a proof-of-concept correction scheme employing a least-squares criterion. As a result, we propose the following constrained linear least-squares formulation:

$$\arg\min_{\hat{\mathbf{m}}} \|\mathbf{P}\hat{\mathbf{m}} - \mathbf{r}\|_2^2, \text{ such that } \mathbf{D}\hat{\mathbf{m}} = \mathbf{d} \text{ and } \mathbf{0} \leq \hat{\mathbf{m}} \leq \mathbf{1}, \tag{S.4}$$

where d is a column vector of length L, defining the radiant emittance of defective pixels, and $\mathbf{D}$ is an $L \times N$ binary matrix specifying the position of defects (i.e., row k of $\mathbf{D}$ has a single non-zero element identifying the defect).

Similar to Lanman et al. [2011], we recognize that Equation S.4 can be efficiently solved using the simultaneous algebraic reconstruction technique (SART) [Andersen and Kak 1984]. Under this framework, the corrected microdisplay image $\hat{\mathbf{m}}^{(k)}$ at iteration k is given by the following update rule:

$$\hat{\mathbf{m}}^{(k)} = \hat{\mathbf{m}}^{(k-1)} - \mathbf{v} \circ (\mathbf{P}^\top(\mathbf{w} \circ (\mathbf{P}\hat{\mathbf{m}}^{(k-1)} - \mathbf{r}))), \tag{S.5}$$

where $\circ$ defines the Hadamard (element-wise) matrix product and the weight vectors $\mathbf{w}$ and $\mathbf{v}$ are defined as follows.

$$w_m = \frac{1}{\sum_{n=1}^{N} P_{mn}} \quad \text{and} \quad v_n = \frac{1}{\sum_{m=1}^{M} P_{mn}} \tag{S.6}$$

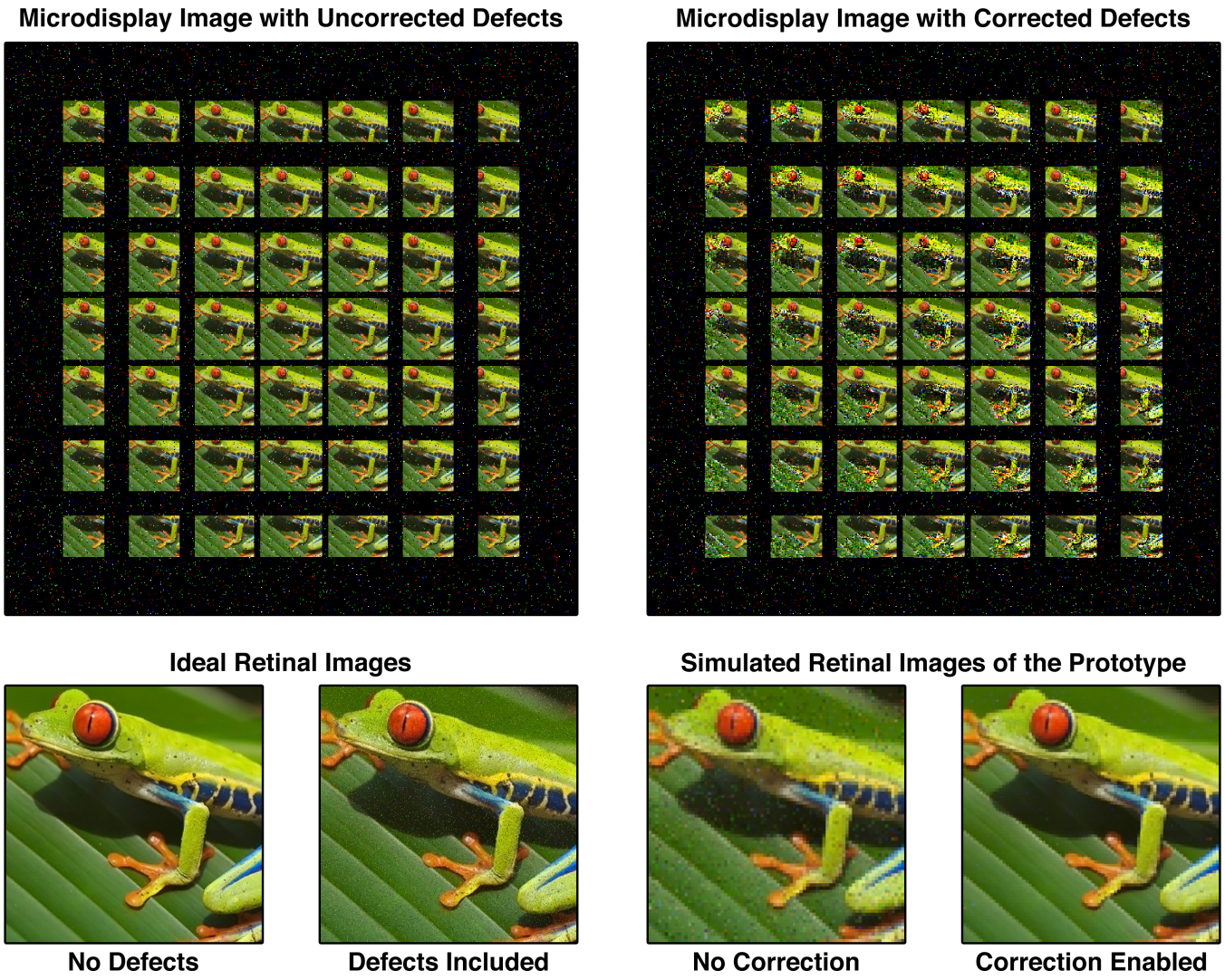

Figure S.14: *Correcting microdisplay defects. Defect densities are 100 times greater than allowed for a "class IV" display, as defined in Table S.1. (Top) Simulated microdisplay images, corresponding with the HMD prototype. Zoom in to inspect alterations made to compensate for defects. (Bottom) Ideal and simulated prototype retinal images.*

Equality and inequality constraints are enforced after each iteration (i.e., fixing and clamping values, respectively). We emphasize that this framework was selected due to the prior demonstration by Lanman et al., in a different application context, that such update rules can be implemented using GPUs for real-time optimization on high-resolution images. However, our current solver comprises a Matlab-based implementation.

Figures S.13 and S.14 include simulated defect correction results. The bottom left-hand side of each figure illustrates the visual impact of defects on a conventional HMD design using a simple magnifier; without redundancy, no compensation is possible. In contrast, the bottom right-hand side shows that near-eye light field displays are robust to profound defect densities. Although only a proof-of-concept implementation, near-eye light field displays appear particularly well-suited for economical fabrication of large-format microdisplays; even if multi-reticle manufacturing is required, yield and cost can be controlled by exploiting redundancy within element images. Yet, a full development of microdisplay defect correction remains a promising direction for future work. This appendix assumes that eye tracking will be available in order to determine the set of rays impinging on the retina in Equation S.3. Future research may explore alternative optimization criteria localizing corrections to elemental images neighboring a defect—mitigating artifacts that are otherwise introduced for pupil positions not observing the defective element. In addition, simplified heuristic correction schemes may reduce computational overhead.

Supplementary References

ANDERSEN, A., AND KAK, A. 1984. Simultaneous Algebraic Reconstruction Technique (SART): A superior implementation of the ART algorithm. *Ultrasonic Imaging 6*, 1, 81–94.

BOURKE, P., 1999. Calculating stereo pairs. `http://paulbourke.net/stereographics/`.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO), 2000. ISO 13406-2: "Ergonomic requirements for work with visual displays based on flat panels – Part 2: Ergonomic requirements for flat panel displays". `https://en.wikipedia.org/wiki/ISO_13406-2`.

IVES, F. E., 1903. Parallax stereogram and process of making same. U.S. Patent 725,567.

JAIN, A., TRAN, L., KHOSHABEH, R., AND NGUYEN, T. 2011. Efficient stereo-to-multiview synthesis. In *Acoustics, Speech and Signal Processing (ICASSP)*, 889–892.

JIN, C., AND JEONG, H. 2008. Intermediate view synthesis for multi-view 3D displays using belief propagation-based stereo matching. In *Convergence and Hybrid Information Technology (ICCIT)*, vol. 1, 919–924.

LANMAN, D., WETZSTEIN, G., HIRSCH, M., HEIDRICH, W., AND RASKAR, R. 2011. Polarization fields: dynamic light field display using multi-layer LCDs. *ACM Trans. Graph. 30*, 6, 186:1–186:10.

LEVOY, M., AND HANRAHAN, P. 1996. Light field rendering. ACM SIGGRAPH, 31–42.

LIPPMANN, G. 1908. Épreuves réversibles donnant la sensation du relief. *Journal of Physics 7*, 4, 821–825.

PAMPLONA, V. F., OLIVEIRA, M. M., ALIAGA, D. G., AND RASKAR, R. 2012. Tailored displays to compensate for visual aberrations. *ACM Trans. Graph. 31*, 4, 81:1–81:12.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. OptiX: a general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4, 66:1–66:13.

RAMACHANDRAN, G., AND RUPP, M. 2012. Multiview synthesis from stereo views. In *Systems, Signals, and Image Processing (IWSSIP)*, 341–345.

SHIBATA, T., KIM, J., HOFFMAN, D. M., AND BANKS, M. S. 2011. The zone of comfort: Predicting visual discomfort with stereo displays. *Journal of Vision 11*, 8.