Supplementary Material:
Focus 3D: Compressive Accommodation Display

In this document we provide additional results and implementation details in support of the primary text. Appendix A presents additional analysis of decomposed masks patterns and the results of a wide field of view display simulation. Appendix B provides details of the prototype display construction and pseudocode for the utilized GPU-based nonnegative tensor factorization (NTF) solver and supporting functions.

A Extended Results

A.1 Simulations of a Wide Tracked Field of View

Figure 15 in the primary text demonstrates motion parallax on our Focus 3D prototype. However, as noted in Section 5 of the primary text, the field of view was only moderate and was limited by the distortions of the inexpensive lens chosen for the prototype. In Figure S.1, we provide monoscopic wide field of view simulations for a tracked user under an identical optical configuration, but with the assumption of a high quality lens. The simulations demonstration focusablility over a wide 56° field of view (136 cm laterally) for a user 127 cm from the display using six time-multiplexed frames.

A.2 Decomposition of Light Field into Time-Multiplexed Masks

Section 3.2 and Figure 6 of the primary text illustrate how NTF exploits correlation between multiple views, allowing a light field with N distinct views to be compressed into fewer than N time-multiplexed frames. Figure S.2 provides an additional light field mask decomposition to help the reader gain insight into the compressive nature of NTF. The figure presents the masks corresponding to the light field displayed in Figure 3 of the primary text, which consists of two sets of 5×5 views placed eye distance (6.4 cm) apart. These 50 views are compressed into 12 pairs of mask patterns that are displayed in sequence on the front LCD layer and backlight. In the right hand column of Figure S.2, each backlight pixel corresponds to one of 50 views. Note that in the time-multiplexed frames, most of the backlight pixels have been illuminated to some degree – indicating that the NTF compression exploits correlation between both the closely spaced views for each eye and the views between eyes – resulting in brighter images than if the 50 views were simply displayed in sequence and illuminated by a single backlight pixel at a time. As shown in Figure 12 of the primary text, the compressed light field has enough fidelity to allow a camera to focus at different depths within the scene.

B Implementation Details

B.1 Additional Details on the GPU-based NTF Implementation with Refractive Element

This section documents our GPU-based implementation of nonnegative tensor factorization (NTF) for Focus 3D, which is based on the NTF of Wetzstein et al. [2012]. As explained in Section 4.2 of the primary text, most of the functions map to the fixed graphics pipeline. NTF is implemented using OpenGL and a set of CG/GLSL shaders; pseudocode is listed below. We assume the Focus 3D display consists of L light-attenuating layers, each displaying F frames in rapid succession, and a backlight behind a lens. The original light field is assumed to consist of V different views. As the display of the decomposed layers is a time-critical operation – requiring a frame rate that matches the monitor refresh rate – our online solver is implemented with two modes: one which uses the GPU to factor the light

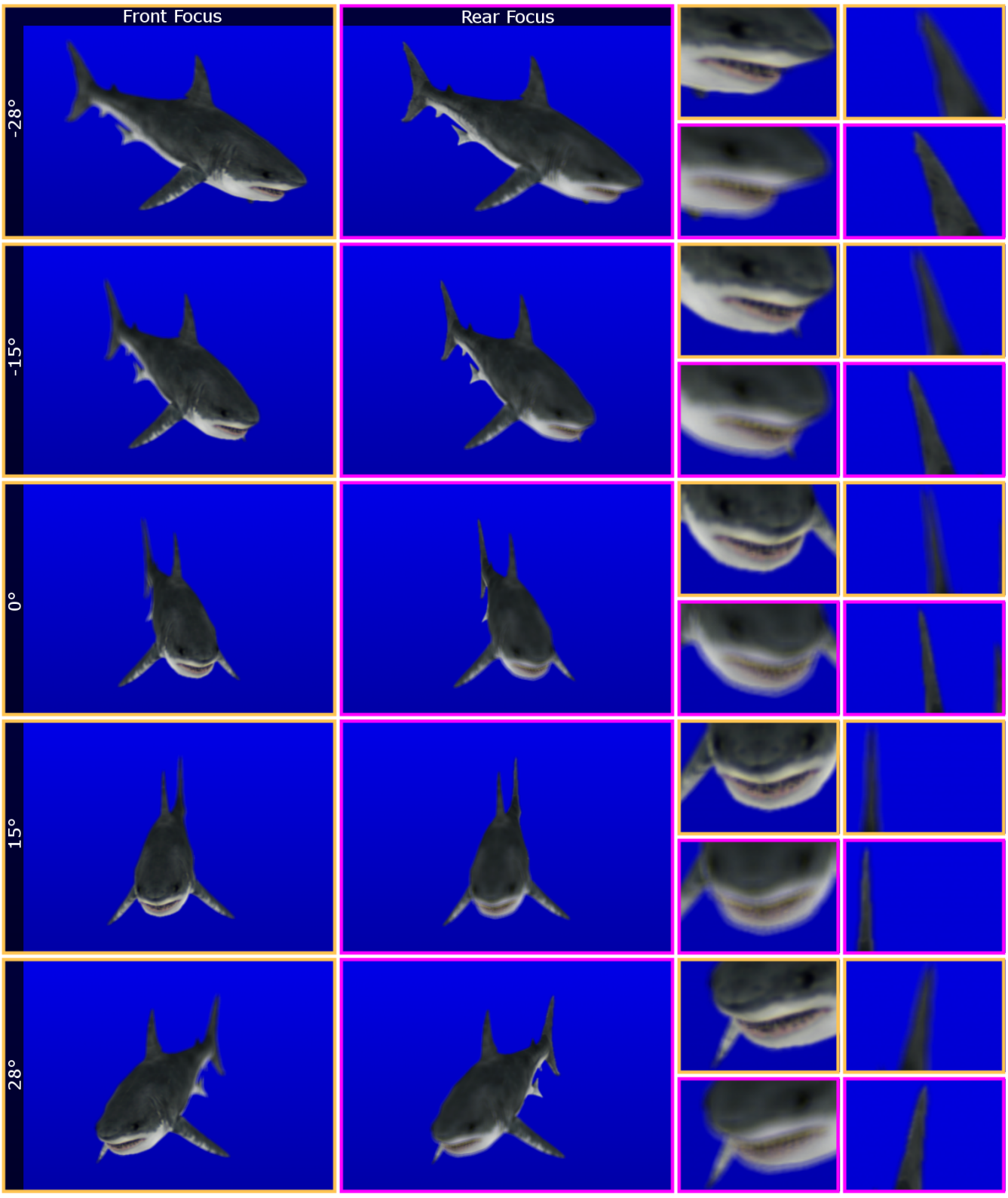

Figure S.1: *Simulation of a wide field of view display with correct accommodation for a tracked user. Five viewpoints, laterally shifted parallel to the display, are shown in the rows while the left and center columns show the front and rear of the shark in focus, respectively. Shark model by artist "wibarra88" of Turbosquid.com.*

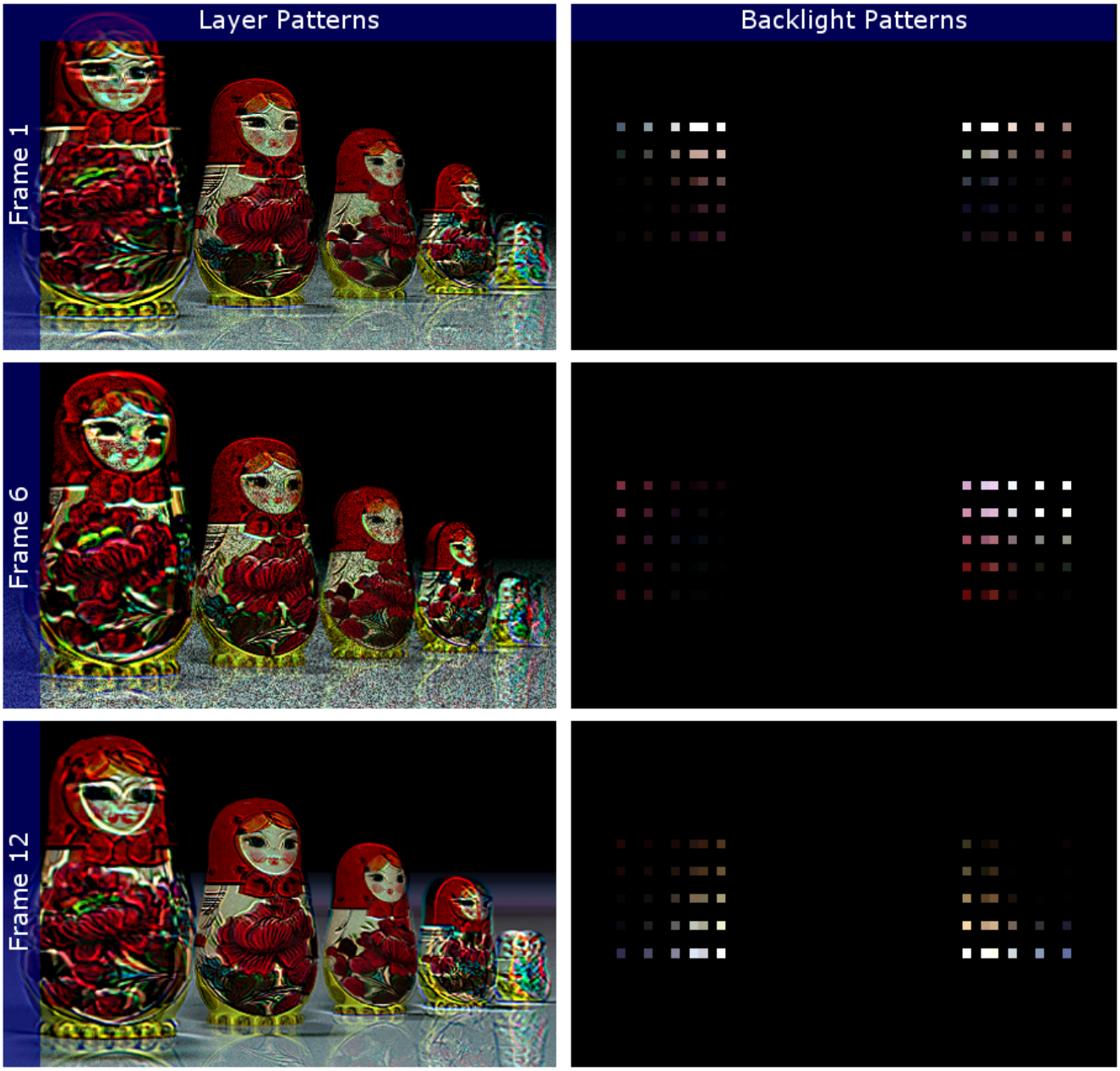

Figure S.2: *Mask patterns for a light field supporting correct accommodation in each eye. Left column: mask patterns displayed on the front LCD layer. Right column: backlight patterns. Rows: Three frames from the 12 frame time-multiplexed sequence. Matryoshka doll model by artist "coboide" of Turbosquid.com.*

field tensor, displaying the solved frames slowly, and a second which disables the iterative optimization and displays solved frames in quick succession. Future implementations might use multiple GPUs or dedicated computational hardware to perform these steps simultaneously.

The decomposition routines and supporting functions are documented on the following pages, implementing weighted nonnegative tensor factorization, as discussed in Section 3.1.2 of the primary text and in greater detail in Wetzstein et al. [2012]. These routines are followed by the main display loop for synchronized rendering of the temporally-multiplexed layers and backlight with the monitor refresh rates.

Algorithm NTF - Content-Updating Thread

```
variables FBO_LF[V], FBO_LF_REC[V], FBO_LAYERS[L][F], FBO_LF_TMP[V], FBO_LAYER_TMP[2], FBO_TMP[2]
variables FBO_BACKLIGHT[F], FBO_BACKLIGHT_TMP[2], FBO_TMP[2]

function threadDisplayLoop()
    // draw light field
    for all light field views v
        activate FBO_LF[v]
        set perspective v
        drawScene(); // render desired 3D scene (e.g., a teapot)
    end
    // factorize light field using NTF
    NTF();
end

function NTF()
    for all iterations i
        // update the layers
        for all layers l
            // draw current estimate of LF into rec buffers
            drawLightFieldFromLayersRec();
            for all frames f
                // draw layers into LF tmp buffers, but leave out current layer
                drawLightFieldFromLayersTmp(l);
                // compute numerator for multiplicative NTF update
                activate FBO_LAYER_TMP[1]
                activate accumulation buffer
                for all light field views v
                    set perspective v as projective texture matrix
                    activate CG_SHADER_MULT2TEXTURES_AND_PROJECTIVE_TEXMAPTHEM( FBO_LF[v], FBO_LF_TMP[v] )
                    draw 2D quad
                end
                deactivate FBO_LAYER_TMP[1]
                // compute denominator for multiplicative NTF update
                activate FBO_LAYER_TMP[2]
                activate accumulation buffer
                for all light field views v
                    set perspective v as projective texture matrix
                    activate CG_SHADER_MULT2TEXTURES_AND_PROJECTIVE_TEXMAPTHEM( FBO_LF_REC[v], FBO_LF_TMP[v] )
                    draw 2D quad
                end
                deactivate FBO_LAYER_TMP[2]
                // update current layer for current frame
                activate FBO_LAYERS[l][f]
                activate CG_SHADER_MULT2TEXTURES_DIVIDEBYOTHER ( FBO_LAYERS[l][f], FBO_LAYER_TMP[1], FBO_LAYER_TMP[2] )
                draw 2D quad
                deactivate FBO_LAYERS[l][f]
            end
        end
        // update the backlight
        // draw current estimate of LF into rec buffers
        drawLightFieldFromLayersRec();
        for all frames f
            // draw layers into LF tmp buffers, but leave out backlight
            drawLightFieldFromLayersTmp(-1);
            // compute numerator for multiplicative NTF update
            for all light field views v
                activate FBO_TMP[1]
                set perspective v as projective texture matrix for lens position
                activate CG_SHADER_MULT2TEXTURES_AND_PROJECTIVE_TEXMAPTHEM( FBO_LF[v], FBO_LF_TMP[v] )
                draw 2D quad
                deactivate FBO_TMP[1]
                //refract from lens onto backlight
                refractReverse (FBO_TMP[1],FBO_TMP[2],v)
                activate accumulation buffer
                activate FBO_BACKLIGHT_TMP[1]
                bind FBO_TMP[2]
                draw 2D quad
                deactivate FBO_BACKLIGHT_TMP[1]
            end
            // compute denominator for multiplicative NTF update
            for all light field views v
                activate FBO_TMP[1]
                set perspective v as projective texture matrix for lens position
                activate CG_SHADER_MULT2TEXTURES_AND_PROJECTIVE_TEXMAPTHEM( FBO_LF_REC[v], FBO_LF_TMP[v] )
                draw 2D quad
                deactivate FBO_TMP[1]
                //refract from lens onto backlight
                refractReverse (FBO_TMP[1],FBO_TMP[2],v)
                activate accumulation buffer
                activate FBO_BACKLIGHT_TMP[2]
                bind FBO_TMP[2]
                draw 2D quad
                deactivate FBO_BACKLIGHT_TMP[2]
            end
            // update current layer for current frame
            activate FBO_BACKLIGHT[f]
            activate CG_SHADER_MULT2TEXTURES_DIVIDEBYOTHER ( FBO_BACKLIGHT[f], FBO_BACKLIGHT_TMP[1], FBO_BACKLIGHT_TMP[2] )
            draw 2D quad
            deactivate FBO_BACKLIGHT[f]
        end
    end
end
```

Algorithm NTF - Additional Helper Functions

```
function drawLightFieldFromLayersRec()
    convertAllLayersAndBacklightToLOG();
    for all views v
        for all frames f
            activate FBO_TMP
            activate accumulation buffer
            for all layers l
                draw layer l, textured with FBO_LAYERS[l][f]
            end
            refractForward(FBO_BACKLIGHT[f], FBO_TMP2,v)
            draw backlight projected onto lens, textured with FBO_TMP2
            deactivate FBO_TMP
            activate FBO_LF_REC[v]
            activate accumulation buffer
            activate CG_SHADER_DRAW_EXPONENTIAL_TEXTURE ( FBO_TMP )
            deactivate FBO_LF_REC[v]
        end
    end
    convertAllLayersAndBacklightFromLOG();
end

function drawLightFieldFromLayersTmp(int leaveOutLayerX)
    convertAllLayersAndBacklightToLOG();
    for all views v
        for all frames f
            activate FBO_TMP
            activate accumulation buffer
            for all layers l
                if leaveOutLayerX!=l
                    draw layer l, textured with FBO_LAYERS[l][f]
                end
            end
            if (leaveOutLayerX!=-1)
                refractForward(FBO_BACKLIGHT[f], FBO_TEMP2,v)
                draw backlight projected onto lens, textured with FBO_TEMP2
            end
            deactivate FBO_TMP
            activate FBO_LF_TMP[v]
            activate accumulation buffer
            activate CG_SHADER_DRAW_EXPONENTIAL_TEXTURE ( FBO_TMP )
            deactivate FBO_LF_TMP[v]
        end
    end
    convertAllLayersAndBacklightFromLOG();
end
```

Algorithm NTF - Lens Refraction Functions

function refractForward(FBO backlightFBO, FBO lensFBO, float eyePos[3])
    for each pixel $p1$ on $lensFBO$
        $r1$ = ray from $eyePos$ to pixel $p1$
        $r2$ = ray $r1$ refracted through lens
        $p2$ = pixel on $backlightFBO$ that intersects with $r2$
        $lensFBO[p1] = backlightFBO[p2]$
    end
end

function refractReverse(FBO lensFBO,FBO backlightFBO, float eyePos[3])
    clear $backlightFBO$
    for each pixel $p1$ on $lensFBO$
        $r1$ = ray from $eyePos$ to pixel $p1$
        $r2$ = ray $r1$ refracted through lens
        $p2$ = pixel on $backlightFBO$ that intersects with $r2$
        $backlightFBO[p2] = backlightFBO[p2] + lensFBO[p1]$
    end
end

Algorithm NTF - Main Display Routine

variables FBO_LAYERS[L][F], FBO_BACKLIGHT[F]

function mainDisplayLoop()
    // draw layers of current frame
    for all layers l
        set viewport for l
        activate FBO_LAYERS[l][f]
        draw textured 2D quad
    end
    // draw backlight of current frame
    set viewport for backlight
    activate FBO_BACKLIGHT[f]
    draw textured 2D quad
    // cycle through frames
    f = ($f < F$) ? f+1 : 0;
end

B.2 Hardware

In this section we provide further details about the physical construction of the prototype Focus 3D display. Refer to Figure S.3 for imagery corresponding to the following description.

Our challenge is to align multiple light modulating layers (LCD panels) and a large Fresnel lens over a relatively long distance. We use *computer numerical control* (CNC) machines to construct the parts required for our hardware.

After removing the LCD panels from their backlight units, we remove the diffusing polarizing film from the front of each panel, making the LCD suitable for image formation. Acetone is used to remove residual adhesive left on the surface of the panel. A transparent polarizing film is then secured in place of the removed polarizing film to restore the intensity modulation capability of the panel.

The LCD panels are mounted on waterjet cut aluminum back planes. We similarly mount the Fresnel lens inside an acrylic sheet, cut to fit on a lasercutter. Each of these layers is placed on a rail cage and accurately spaced using lasercut plastic clips. Our rail cage is assembled from waterjet cut $\frac{3}{8}$" aluminum posts, fitted to a wooden base cut to size on a ShopBot CNC mill. Machined alignment holes are then match drilled to ensure the posts and layers slide smoothly on $\frac{1}{4}$" steel rails.

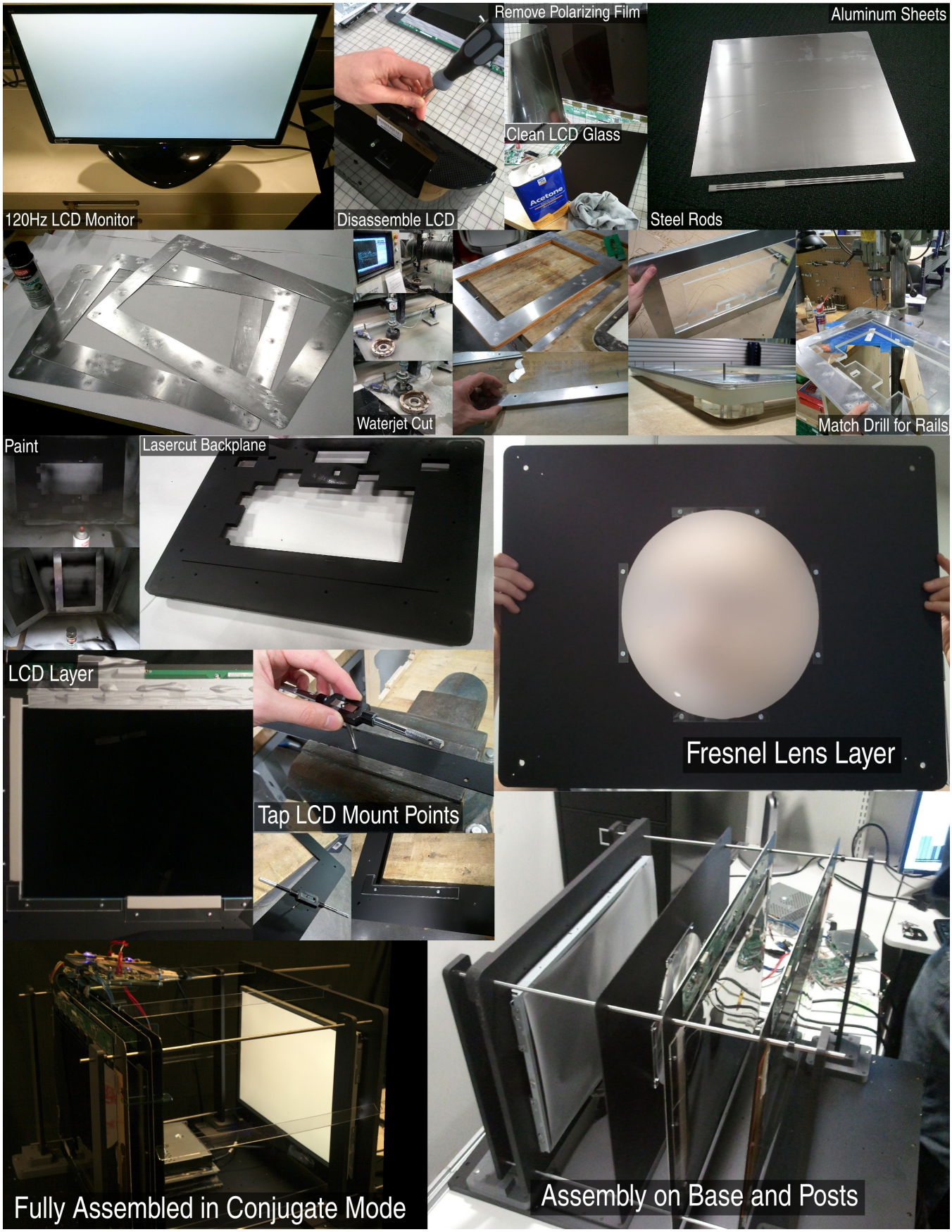

Figure S.3: *Prototype assembly montage. Backlight and polarizing films are removed from LCD panels. Aluminum frames and steel rail system are assembled. LCDs and Fresnel lens are placed on rails for alignment.*

Supplementary References

WETZSTEIN, G., LANMAN, D., HIRSCH, M., AND RASKAR, R. 2012. Tensor Displays: Compressive Light Field Synthesis using Multilayer Displays with Directional Backlighting. *ACM Trans. Graph. (SIGGRAPH) 31*, 1–11.