Supplementary Appendices

Cascaded Displays: Spatiotemporal Superresolution using Offset Pixel Layers

Felix Heide Douglas Lanman Dikpal Reddy Jan Kautz Kari Pulli David Luebke
NVIDIA Research

This document contains additional results and analysis in support of the primary text.

A Weighted Nonnegative Matrix Factorization

Let us first again formulate the weighted nonnegative matrix factorization problem (WNMF) which we are trying solve for the various spatial superresolution application in the paper. Given a non-negative matrix $\mathbf{T} \in \mathbb{R}_+^{m \times n}$, and a target rank $r < \min(m,n)$, we want to solve:

$$\begin{aligned} \mathbf{A}_{\text{opt}}, \mathbf{B}_{\text{opt}} &= \underset{\mathbf{A} \in \mathbb{R}_+^{m \times r}, \mathbf{B} \in \mathbb{R}_+^{n \times r}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{T} - \mathbf{A}\mathbf{B}^T \right\|_{\mathbf{W}}^2 \\ &= \underset{\mathbf{A} \in \mathbb{R}_+^{m \times r}, \mathbf{B} \in \mathbb{R}_+^{n \times r}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{W} \circ \mathbf{T} - \mathbf{W} \circ \mathbf{A}\mathbf{B}^T \right\|_F^2 \end{aligned} \tag{S.1}$$

A.1 A Comparison of Weighted NMF Algorithms for Cascaded Display Factorization

A few efficient algorithms for weighted NMF are known. In this work, we consider the weighted multiplicative update rules by Blondel et al. [2008], the weighted rank-one residue iteration (WRRI) method [Ho 2008], and the alternating least-squares Newton (ALS-Newton) method [Paatero and Tapper 1994]. In the following we we analyze their convergence, their numerical stability and their runtime on a recent GPU.

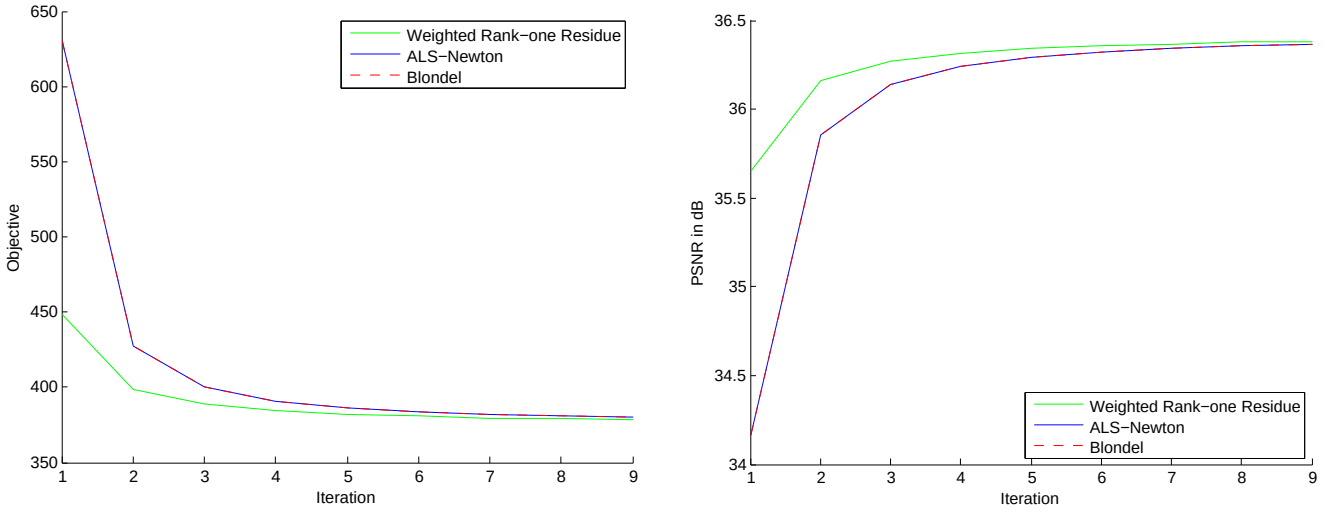

Figure S.1: *Convergence with **double precision** factorization (rank-1 factorization on* 1576×1050 *target image). Left: Objective function, Right: PSNR for comparison.*

In Figure S.1 we use each algorithm to factorize a target HD image (1576×1050 pixels) into a rank-1 dual-layer representation, as discussed in Section 3 in the paper. In this case, we ran each algorithm using double precision floating point numbers. As can be seen, all three methods achieve similar results after a few iterations. We note though that WRRI achieves better quality when few iterations are applied.

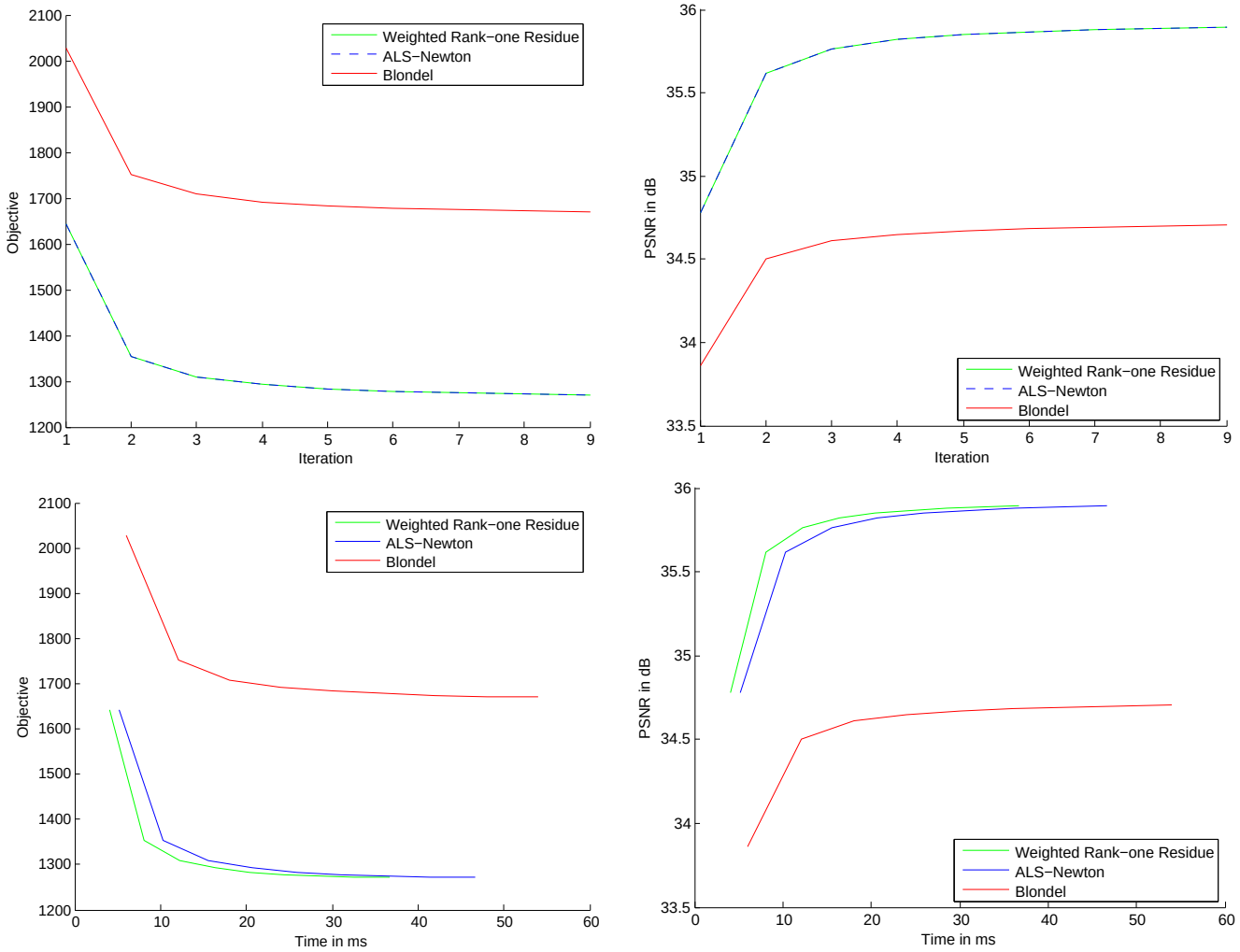

Figure S.2: *Convergence with **single precision** factorization (rank-1 factorization on* 1576×1050 *target image). Left: Objective function, Right: PSNR for comparison. Top: Iterations. Bottom: GPU Timings.*

In Figure S.2 we show the convergence of the three algorithms when using floating point precision. As is evident Blondel et al.'s update rules are numerically less stable than WRRI and ALS-Newton. Furthermore, we have implemented all three methods on the GPU (Geforce GTX 770M) to compare actual run-time, which matters more than pure convergence rate (bottom of figure). WRRI produces better factorizations in less time compared to the other two methods. It is the fastest due to fewer required memory accesses ($2\times$ less than the other methods). As for Blondel et al.'s method, we found images, where it never produced visually acceptable results. We note that in this example ALS-Newton is quite fast for rank-1, which is predominantly because we have adapted it to our specific problem of for rank-1 factorizations (see Section A.3).

The following table lists the performance we achieve when running three iterations with each method for a 1576×1050 frames (timings averaged over 10 frames):

Method	Newton	WRRI	Blondel
Time in [ms]	15.554	**12.256**	18.053
FPS	64.3	**81.6**	55.4

As can be seen, WRRI is certainly fast enough to be used in real-time applications, such as interactive content or video playback.

A.2 Weighted Nonnegative Matrix Factorization for *Joint* Spatiotemporal Superresolution

Let us formulate the general spatio-temporal superresolution optimization problem. If we stack every pixel value at every staggered refresh time in a large vector for each layer, we can model the spatio-temporal layer reconstruction again as a weighted rank-1 NMF problem.

As in the previous section, assume we are given a non-negative matrix $\mathbf{T} \in \mathbb{R}_+^{m \times n}$. Then we want to solve:

$$\begin{aligned} \mathbf{a}_{\text{opt}}, \mathbf{b}_{\text{opt}} &= \underset{\mathbf{a} \in \mathbb{R}_+^m, \mathbf{b} \in \mathbb{R}_+^n}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{T} - \mathbf{C}\mathbf{P}_1 \mathbf{a}\mathbf{b}^T \mathbf{P}_2 \right\|_{\mathbf{W}}^2 \\ &= \underset{\mathbf{a} \in \mathbb{R}_+^m, \mathbf{b} \in \mathbb{R}_+^n}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{W} \circ \mathbf{T} - \mathbf{W} \circ \mathbf{C}\mathbf{P}_1 \mathbf{a}\mathbf{b}^T \mathbf{P}_2 \right\|_F^2 \end{aligned} \tag{S.2}$$

The vectors $\mathbf{a}, \mathbf{b}$ are very large here, as they contain all layer pixels over all timesteps. The matrices $\mathbf{P}_1, \mathbf{P}_2$ are permutation matrices, where $\mathbf{P}_1$ will permute the rows of the $\mathbf{a}\mathbf{b}^T$ which contains all possible spatial and temporal layer interactions (forward and backward in time). The matrix $\mathbf{P}_2$ will permute the columns of this matrix. Together they permute $\mathbf{a}\mathbf{b}^T$, so that the resulting matrix contains the stacked image corresponding to a particular time-step in one column. The weight matrix $\mathbf{W}$ assigns 0 weight to the large parts of this matrix, which correspond to no layer interaction. Finally, the matrix $\mathbf{C}$ is a potential blur applied to the superresolved image (e.g., a diffuser). A small blur allows an additive spatial coupling of nearby pixels.

After describing the spatio-temporal optimization problem ((S.2)), we now derive matrix factorization update rules. For simplicity we do this using the well-known multiplicative Lee and Seung NMF rules [Lee and Seung 1999], including the weight-adaption by Blondel et al. [2008]; however our derivations apply to other NMF algorithms straightforwardly. The Blondel et al. NMF rules for Eq. (S.1) were:

$$\mathbf{B} \leftarrow \mathbf{B} \circ \frac{(\mathbf{W} \circ \mathbf{T})^T \mathbf{A}}{(\mathbf{W} \circ \mathbf{A}\mathbf{B}^T)^T \mathbf{A}}, \quad \mathbf{A} \leftarrow \mathbf{A} \circ \frac{(\mathbf{W} \circ \mathbf{T})\mathbf{B}}{(\mathbf{W} \circ \mathbf{A}\mathbf{B}^T)\mathbf{B}}. \tag{S.3}$$

where the double lines ($\frac{\circ}{=}$) denote elementwise division. After deriving the corresponding rules for (S.2) by following the same approach as Ho [Ho 2008], we found that for our generalization of the NMF problem we can use the following simpler derivation. Let us substitute

$$\begin{aligned} \mathbf{A} &:= \mathbf{C}\mathbf{P}_1 \mathbf{a} \\ \mathbf{B} &:= \left(\mathbf{b}^T \mathbf{P}_2\right)^T = \mathbf{P}_2^T \mathbf{b} \end{aligned} \tag{S.4}$$

Then we get for the first update of Eq. (S.3):

$$\begin{aligned} \mathbf{B} = \mathbf{P}_2^T \mathbf{b} &\leftarrow \mathbf{P}_2^T \mathbf{b} \circ \frac{(\mathbf{W} \circ \mathbf{T})^T (\mathbf{C}\mathbf{P}_1 \mathbf{a})}{(\mathbf{W} \circ \mathbf{C}\mathbf{P}_1 \mathbf{a}\mathbf{b}^T \mathbf{P}_2)^T (\mathbf{C}\mathbf{P}_1 \mathbf{a})} \\ \Leftrightarrow \mathbf{P}_2 \mathbf{P}_2^T \mathbf{b} &\leftarrow \mathbf{P}_2 \mathbf{P}_2^T \mathbf{b} \circ \frac{\mathbf{P}_2 (\mathbf{W} \circ \mathbf{T})^T (\mathbf{C}\mathbf{P}_1 \mathbf{a})}{\mathbf{P}_2 (\mathbf{W} \circ \mathbf{C}\mathbf{P}_1 \mathbf{a}\mathbf{b}^T \mathbf{P}_2)^T (\mathbf{C}\mathbf{P}_1 \mathbf{a})} \\ \Leftrightarrow \mathbf{b} &\leftarrow \mathbf{b} \circ \frac{\mathbf{P}_2 (\mathbf{W} \circ \mathbf{T})^T (\mathbf{C}\mathbf{P}_1 \mathbf{a})}{\mathbf{P}_2 (\mathbf{W} \circ \mathbf{C}\mathbf{P}_1 \mathbf{a}\mathbf{b}^T \mathbf{P}_2)^T (\mathbf{C}\mathbf{P}_1 \mathbf{a})} \\ \Leftrightarrow \mathbf{b} &\leftarrow \mathbf{b} \circ \frac{(\mathbf{P}_1^T \mathbf{C}^T (\mathbf{W} \circ \mathbf{T}) \mathbf{P}_2^T)^T \mathbf{a}}{(\mathbf{P}_1^T \mathbf{C}^T (\mathbf{W} \circ \mathbf{C}\mathbf{P}_1 \mathbf{a}\mathbf{b}^T \mathbf{P}_2) \mathbf{P}_2^T)^T \mathbf{a}} \end{aligned} \tag{S.5}$$

Line three follows because permutations matrices have the property $\mathbf{P}^{-1} = \mathbf{P}^T$. The last line shows that the update can be computed efficiently in parallel. It is very similar to the updates from (S.3). The update for $\mathbf{a}$ follows from

symmetry:

$$\mathbf{a} \leftarrow \mathbf{a} \circ \frac{(\mathbf{P}_1^T \mathbf{C}^T (\mathbf{W} \circ \mathbf{T}) \mathbf{P}_2^T) \mathbf{b}}{(\mathbf{P}_1^T \mathbf{C}^T (\mathbf{W} \circ \mathbf{C}\mathbf{P}_1 \mathbf{a}\mathbf{b}^T \mathbf{P}_2) \mathbf{P}_2^T) \mathbf{b}} \tag{S.6}$$

Our derivation using (S.4) can be applied analogously to the WRRI update-rules.

A.3 Real-Time Rank-1 Factorizations using an Alternating Least Squares using Newton

As discussed by Ho [2008], the alternating least squares method for non-negative matrix factorization is rarely used, because it is inefficient. In contrast, the more recent WRRI is very efficient, which is why it is used in our paper. However, we have discovered that for our specific problem and for rank-1 factorizations, there is an interesting way to speed up ALS-Newton considerably.

For rank $r = 1$, our general nonnegative matrix factorization problem from Eq. (S.1) simplifies to:

$$\mathbf{a}_{\text{opt}}, \mathbf{b}_{\text{opt}} = \underset{\mathbf{a}\in\mathbb{R}_+^m, \mathbf{b}\in\mathbb{R}_+^n}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{T} - \mathbf{a}\mathbf{b}^T \right\|_{\mathbf{W}}^2 \tag{S.7}$$

In an alternating least squares scheme, one solves the biconvex problem from above by alternately solving for one of the two variables $\mathbf{a}, \mathbf{b}$ while fixing the other one and iterating (Algorithm 1).

Algorithm 1 Rank-1 Alternating Least Squares weighted NMF

1: $k = 0$, $\mathbf{a}_{\text{opt}}^0 = \mathbf{a}_{init}, \mathbf{b}_{\text{opt}}^0 = \mathbf{b}_{init}$
2: **repeat**
3: $\quad \mathbf{b}_{\text{opt}}^{k+1} := \underset{\mathbf{b}\in\mathbb{R}_+^n}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{T} - \mathbf{a}\mathbf{b}^T \right\|_{\mathbf{W}}^2$ ▷ **b-step**
4: $\quad \mathbf{a}_{\text{opt}}^{k+1} := \underset{\mathbf{a}\in\mathbb{R}_+^m}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{T} - \mathbf{a}\mathbf{b}^T \right\|_{\mathbf{W}}^2$ ▷ **a-step**
5: $\quad k := k + 1$
6: **until** Optimality achieved

We note that for our case $r = 1$, we can actually remove the non-negativity constraint (i.e., $\mathbf{b} \in \mathbb{R}_+^n$ and $\mathbf{a} \in \mathbb{R}_+^m$) in step 3 and 4. The trick is that after solving the unconstrained (and hence convex) subproblems of Algorithm 1, we can always project the solution to a non-negative solution with the same objective function value (or better) by simply flipping the sign of the negative elements (assuming that the previous solution does not harm the constraint as well). So we get the algorithm shown in Algorithm 2.

Algorithm 2 Unconstrainted Rank-1 Alternating Least Squares weighted NMF

1: $k = 0$, $\mathbf{a}_{\text{opt}}^0 = \mathbf{a}_{init}, \mathbf{b}_{\text{opt}}^0 = \mathbf{b}_{init}$
2: **repeat**
3: $\quad \mathbf{b}_{\text{opt}}^{k+1} := \underset{\mathbf{b}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{T} - \mathbf{a}\mathbf{b}^T \right\|_{\mathbf{W}}^2$ ▷ **b-step**
4: $\quad \mathbf{b}_{\text{opt}}^{k+1} := \operatorname{sign}\left(\mathbf{b}_{\text{opt}}^{k+1}\right) \circ \mathbf{b}_{\text{opt}}^{k+1}$
5: $\quad \mathbf{a}_{\text{opt}}^{k+1} := \underset{\mathbf{a}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{T} - \mathbf{a}\mathbf{b}^T \right\|_{\mathbf{W}}^2$ ▷ **a-step**
6: $\quad \mathbf{a}_{\text{opt}}^{k+1} := \operatorname{sign}\left(\mathbf{a}_{\text{opt}}^{k+1}\right) \circ \mathbf{a}_{\text{opt}}^{k+1}$
7: $\quad k := k + 1$
8: **until** Optimality achieved

We have formulated our non-convex problem now as a sequence of convex optimization problems. Let us consider the **b-step** in algorithm 2; the derivations for **a-step** follow from symmetry. We solve the **b-step** using Newton's

method which has quadratic convergence. For that we derive the gradient and Hessian of $f(\mathbf{b})$ with:

$$\begin{aligned}
\mathbf{b}_{\text{opt}} &= \underset{\mathbf{b}}{\text{argmin}} \; \frac{1}{2} \left\| \mathbf{T} - \mathbf{a}\mathbf{b}^T \right\|_{\mathbf{W}}^2 \\
&= \underset{\mathbf{b}}{\text{argmin}} \; \frac{1}{2} \left\| \mathbf{D}_{\mathbf{W}}\mathbf{t} - \mathbf{D}_{\mathbf{W}}\mathbf{O}_{\mathbf{a}}\mathbf{b} \right\|_F^2 \\
&= \underset{\mathbf{b}}{\text{argmin}} \; \frac{1}{2} \left(\mathbf{t}^T\mathbf{D}_{\mathbf{W}}^T\mathbf{D}_{\mathbf{W}}\mathbf{t} - 2\mathbf{t}^T\mathbf{D}_{\mathbf{W}}^T\mathbf{D}_{\mathbf{W}}\mathbf{O}_{\mathbf{a}}\mathbf{b} + \mathbf{O}_{\mathbf{a}}^T\mathbf{D}_{\mathbf{W}}\mathbf{O}_{\mathbf{a}}\mathbf{b} \right) \\
&= \underset{\mathbf{b}}{\text{argmin}} \; \underbrace{\frac{1}{2} \left(\mathbf{t}^T\mathbf{D}_{\mathbf{W}}^2\mathbf{t} - 2\mathbf{t}^T\mathbf{D}_{\mathbf{W}}^2\mathbf{O}_{\mathbf{a}}\mathbf{b} + \mathbf{O}_{\mathbf{a}}^T\mathbf{D}_{\mathbf{W}}\mathbf{O}_{\mathbf{a}}\mathbf{b} \right)}_{f(\mathbf{b})}
\end{aligned} \tag{S.8}$$

where in line two we have introduced the matrices $\mathbf{D}_{(\cdot)}$, which puts the matrix from the subscript on the diagonal and the matrix $\mathbf{O}_{(\cdot)}$, which corresponds to the outer vector product operation with the vector in the subscript and the rhs, followed by vectorization. The second line allows us to remove the Frobenius norm and we can now easily derive the gradient and Hessian of f. For the gradient we get

$$\begin{aligned}
\nabla f &= \mathbf{O}_{\mathbf{a}}^T\mathbf{D}_{\mathbf{W}}\mathbf{O}_{\mathbf{a}}\mathbf{b} - \mathbf{O}_{\mathbf{a}}^T\mathbf{D}_{\mathbf{W}}^2\mathbf{t} \\
&= \mathbf{O}_{\mathbf{a}}^T\mathbf{D}_{\mathbf{W}\circ(\mathbf{a}\mathbf{b}^T)-\mathbf{W}\circ\mathbf{W}\circ\mathbf{T}}\mathbf{1}
\end{aligned} \tag{S.9}$$

The operator $\mathbf{O}_{\mathbf{a}}^T$ is the same as the outer vector product operation plus subsequent summation over the rows of the resulting matrix. So we simply need to do the pointwise operation $\mathbf{W} \circ \mathbf{a}\mathbf{b}^T - \mathbf{W} \circ \mathbf{W} \circ \mathbf{T}$, do the outer product with $\mathbf{a}$, sum over the rows of the corresponding matrix, which yields then the gradient w.r.t. $\mathbf{b}$. For the Hessian we get a diagonal matrix with

$$\begin{aligned}
\frac{\partial^2 f}{\partial b^2} &= \mathbf{O}_{\mathbf{a}}^T\mathbf{D}_{\mathbf{W}}\mathbf{O}_{\mathbf{a}} \\
&= \mathbf{O}_{\mathbf{a}}^T\mathbf{D}_{\mathbf{W}\circ(\mathbf{a}\cdot\mathbf{1}^T)}
\end{aligned} \tag{S.10}$$

By using the Newton method we can exploit the structure of our problem. Since the Hessian $H(f) = \mathbf{D}_{\frac{\partial^2 f}{\partial b^2}}$ is a diagonal matrix, the inverse in Newton's method becomes simply a pointwise division and we get Algorithm 3:

Algorithm 3 Full newton for rank 1

1: **repeat**
2: $\quad \mathbf{b}_{\text{opt}}^{k+1} := \mathbf{b}_{\text{opt}}^{k+1} - \frac{\nabla f}{\frac{\partial^2 f}{\partial b^2}}$ ▷ Pointwise division
3: $\quad k := k + 1$
4: **until** Optimality achieved

We now have all the components to implement Algorithm 2. The resulting method is very efficient, but is still slightly slower than WRRI, as demonstrated in Section A.1.

B Real-time CUDA Code for Rank-1 Factorization

We include our CUDA code for real-time rank-1 matrix factorizations. Our timings in Section A.1 were computed with this kernel. The code supports three different update rules, Blondel et al.'s, WRRI, and ALS Newton.

The code itself consists of two kernels, one computes the the nominator (or gradient) and denominator (or Hessian) for an update (for a considered layer) and another one performs the update given those components. Note, how the three methods are surprisingly similar in terms of code.

```
1
2  ////////////////////////////////////////////////////////////////////////////
3  // rank-1 matrix factorization for NMF, WRRI, ALS Newton
4  ////////////////////////////////////////////////////////////////////////////
5
6  //Computes denominator(or hessian) [d_denom] and nominator(or gradient) [d_nom] for update rules for
7  //layer A [d_A] or layer B [d_B] given the fragments (for numCh color channels).
8
9  //The integrated fragment color values [d_samples], their normalized area [d_weights] and
10 //intersection indices on each layer [d_layerInt] are given for the fragments.
11
12 //The kernel supports NMF (method == 0), WRRI (method == 1), NEWTON (method == 2)
13
14 static __global__ void factorization_kernel( float *d_A, float *d_B, int width_layer, int height_layer,↩
       int numCh, float* d_samples, float* d_weights, int numFragments, int* d_layerInt, int ABflag, ↩
      float *d_denom, float* d_nom, int method)
15 {
16    //Vars
17    float denom, nom, a_curr, b_curr, t_curr, w_curr, val ;
18    int layerAIdx, layerBIdx;
19
20    //Parallel over fragments
21    int fch = blockIdx.x * blockDim.x + threadIdx.x;
22    for (; fch < numFragments * numCh; fch += gridDim.x * blockDim.x)
23    {
24       //Indices
25       int f = fch % numFragments;
26       int ch = fch / numFragments;
27
28       //Channel offset
29       int chOffLayer = ch * (width_layer * height_layer);
30
31       //For current fragment extract indices on both layers and the fragments area
32       layerAIdx = d_layerInt[2 * f + 0];
33       layerBIdx = d_layerInt[2 * f + 1];
34
35       //Target image fragment value
36       t_curr = d_samples[fch];
37       a_curr = d_A[chOffLayer +   layerAIdx];
38       b_curr = d_B[chOffLayer +   layerBIdx];
39       w_curr = d_weights[f];
40
41       //Update and accumulate
42       if( ABflag == 0 ) //Update A (ABflag == 0), or update B (ABflag != 0)
43       {
44          if( method == 0 )
45          {
46             //#### NMF
47             denom = (a_curr * b_curr * w_curr) * b_curr; //Denominator wrt A
48             nom = b_curr * (t_curr * w_curr); //Nominator wrt A
49          }
50          else if( method == 1 )
51          {
52             //#### WRRI
53             denom = (b_curr * b_curr) * w_curr; //Denominator wrt A
54             nom = b_curr * (t_curr * w_curr); //Nominator wrt A
55          }
```

```
56          else if( method == 2 )
57          {
58             //#### NEWTON
59             nom = a_curr * (b_curr * b_curr) * w_curr - b_curr *  t_curr * w_curr * w_curr ; //Grad wrt↩
                   A
60             denom = b_curr * b_curr * w_curr; //Hessian wrt A
61          }
62
63          //Accumulate
64          atomicAdd( &(d_denom[chOffLayer +   layerAIdx]), denom);
65          atomicAdd( &(d_nom[chOffLayer +   layerAIdx]), nom);
66       }
67       else
68       {
69          if( method == 0 )
70          {
71             //#### NMF
72             denom = a_curr * (a_curr * b_curr * w_curr); //Denominator wrt B
73             nom = a_curr * (t_curr * w_curr); //Nominator wrt B
74          }
75          else if( method == 1 )
76          {
77             //#### WRRI
78             denom = (a_curr * a_curr) * w_curr; //Denominator wrt B
79             nom = a_curr * (t_curr * w_curr); //Nominator wrt B
80          }
81          else if( method == 2 )
82          {
83             //#### NEWTON
84             nom = (a_curr * a_curr) * b_curr * w_curr - a_curr * t_curr * w_curr * w_curr ; //Grad wrt ↩
                   B
85             denom = a_curr * a_curr * w_curr; //Hessian wrt B
86          }
87
88          //Accumulate
89          atomicAdd( &(d_denom[chOffLayer +   layerBIdx]), denom);
90          atomicAdd( &(d_nom[chOffLayer +   layerBIdx]), nom);
91       }
92
93    }
94 }
95
96
97 //Updates the layers A [d_A] or layer B [d_B] given the previously computed
98 //denominator(or hessian) [d_denom] and nominator(gradient) [or d_nom].
99
100 //The kernel supports NMF (method == 0), WRRI (method == 1), NEWTON (method == 2)
101 //The arrays d_denom and d_nom are reset afterwards.
102
103 static __global__ void update_kernel( float *d_A, float *d_B, int width_layer, int height_layer, int ↩
        numCh, float *d_denom, float* d_nom, int ABflag, int method )
104 {
105    //Vals
106    float val, nom, denom;
107
108    //Parallel over output
109    int xych = blockIdx.x * blockDim.x + threadIdx.x;
110    for (; xych < width_layer * height_layer * numCh; xych += gridDim.x * blockDim.x)
111    {
112
113       //Nom and denom
114       denom = d_denom[xych];
115       nom = d_nom[xych];
116
117          //Get current val and do update
118       if( ABflag == 0 )
119       {
120
121          if( method == 0 )
122          {
```

```
            //#### NMF
            val = d_A[xych];
            d_A[xych] = fminf( fmaxf( val * fmaxf(nom, 1.0E-9) / (denom + 1.0E-9), 0.f), 1.f );
         }
         else if( method == 1 )
         {
            //#### WRRI
            //Write
            if( denom <= 0 )
            {
               d_A[xych] = 0.f;
            }
            else
            {
               d_A[xych] = fminf( fmaxf( fmaxf(nom,0.f) / denom, 0.f), 1.f );
            }
         }
         else if( method == 2 )
         {
            //#### NEWTON
            //Write
            val = d_A[xych];
            d_A[xych] = fminf( fmaxf( val - nom/denom, 0.f), 1.f );
         }

      }
      else
      {

         if( method == 0 )
         {
            //#### NMF
            val = d_B[xych];
            d_B[xych] = fminf( fmaxf( val * fmaxf(nom, 1.0E-9) / (denom + 1.0E-9), 0.f), 1.f );
         }
         else if( method == 1 )
         {
            //#### WRRI
            //Write
            if( denom <= 0 )
            {
               d_B[xych] = 0.f;
            }
            else
            {
               d_B[xych] = fminf( fmaxf( fmaxf(nom,0.f) / denom, 0.f), 1.f );
            }
         }
         else if( method == 2 )
         {
            //#### NEWTON
            //Write
            val = d_B[xych];
            d_B[xych] = fminf( fmaxf( val - nom/denom, 0.f), 1.f );
         }

      }

      //Reset nom and denom
      d_denom[xych] = 0.f;
      d_nom[xych] = 0.f;
   }
}
```

Listing 1: *Real-time matrix factorization code.*

C Nonnegative Tensor Factorization for Multi-Layer Cascaded Displays

Multi-layer cascaded displays require a weighted nonnegative tensor factorization (WNTF). We use multiplicative update rules in this circumstance, generalizing the two-layer update rules given by Equation 4 in the primary text. However, WRRI rules for WNTF are provided by Ho [2008] in his Chapter 4.5. We briefly summarize the multiplicative WNTF rules, as they apply for spatial superresolution with cascaded multi-layer displays (i.e., three or more layers placed in direct contact). We point the interested reader to Ho [2008] for extended discussion of WNTF.

The image formation model presented in Section 3.1 can be generalized to three layers as follows:

$$s_{i_1,i_2,i_3} = \sum_{k=1}^{K} w_{i_1,i_2,i_3} \left(a_{i_1}^{(k)} b_{i_2}^{(k)} c_{i_3}^{(k)} \right), \tag{S.11}$$

where we assume a bottom layer with I_1 pixels, a middle layer with I_2 pixels, and a top layers with I_3 pixels. As in the primary text, K time-multiplexed frames are presented to the viewer at a rate exceeding the critical flicker fusion threshold. The transmissivity of pixel i_3 in the top layer, for frame k, is denoted $c_{i_3}^{(k)}$, such that $0 \leq c_{i_3}^{(k)} \leq 1$. Note that w_{i_1,i_2,i_3} denotes the cumulative overlap of pixels i_1, i_2, and i_3.

At this point, we adopt a tensor representation for our image formation model. We encourage readers to refer to Kolder and Bader [2009] and Wetzstein et al. [2012] for a review of multilinear algebra; in particular, we emphasize that the latter develops a closely-related application of WNTF to multi-layer 3D displays. Similar to these references, we define the *canonical decomposition* of an order-3, rank-K tensor as follows:

$$[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] := \sum_{k=1}^{K} \mathbf{x}_k \star \mathbf{y}_k \star \mathbf{z}_k, \tag{S.12}$$

where $\star$ denotes the vector outer product and $\{\mathbf{x}_k, \mathbf{y}_k, \mathbf{z}_k\}$ represent column k of their respective matrices. Equation S.11 can be used to concisely express image formation by a three-layer cascaded display:

$$\mathcal{S} = \mathcal{W} \circ [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] = \mathcal{W} \circ \left(\sum_{k=1}^{K} \mathbf{a}_k \star \mathbf{b}_k \star \mathbf{c}_k \right), \tag{S.13}$$

where $\mathcal{S}$ is a sparse $I_1 \times I_2 \times I_3$ tensor containing the effective emissivities of the subpixel fragments, $\mathcal{W}$ is also a sparse $I_1 \times I_2 \times I_3$ tensor tabulating the cumulative pixel overlaps, and $\circ$ denotes the Hadamard (elementwise) product. Observe that $\{\mathbf{a}_k, \mathbf{b}_k, \mathbf{c}_k\}$ represent the pixel values displayed on their respective layers during frame k (e.g., in lexicographic order); hence, matrix $\mathbf{A}$ equals the concatenation of the frames displayed on the first layer such that $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_K]$ (similarly for the other layers).

Given this image formation model, we propose the following objective function for optimal three-layer factorizations:

$$\underset{\{\mathbf{0} \leq \mathbf{A} \leq \mathbf{1},\ \mathbf{0} \leq \mathbf{B} \leq \mathbf{1},\ \mathbf{0} \leq \mathbf{C} \leq \mathbf{1}\}}{\arg\min} \frac{1}{2} \left\| \mathcal{W} \circ (\beta \mathcal{T} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) \right\|_2^2. \tag{S.14}$$

Similar to Equation 3 in the primary text, β is the dimming factor applied to the target subpixel fragment emissivities $\mathcal{W} \circ \mathcal{T}$. Following from Wetzstein et al. [2012], this objective can be minimized by application of the following multiplicative update rules.

$$\mathbf{A} \leftarrow \mathbf{A} \circ \left(\frac{(\mathbf{W}_{(1)} \circ (\beta \mathbf{T}_{(1)}))(\mathbf{C} \odot \mathbf{B})}{(\mathbf{W}_{(1)} \circ (\mathbf{A}(\mathbf{C} \odot \mathbf{B})^T))(\mathbf{C} \odot \mathbf{B})} \right) \tag{S.15}$$

$$\mathbf{B} \leftarrow \mathbf{B} \circ \left(\frac{(\mathbf{W}_{(2)} \circ (\beta \mathbf{T}_{(2)}))(\mathbf{C} \odot \mathbf{A})}{(\mathbf{W}_{(2)} \circ (\mathbf{B}(\mathbf{C} \odot \mathbf{A})^T))(\mathbf{C} \odot \mathbf{A})} \right) \tag{S.16}$$

$$\mathbf{C} \leftarrow \mathbf{C} \circ \left(\frac{(\mathbf{W}_{(3)} \circ (\beta \mathbf{T}_{(3)}))(\mathbf{B} \odot \mathbf{A})}{(\mathbf{W}_{(3)} \circ (\mathbf{C}(\mathbf{B} \odot \mathbf{A})^T))(\mathbf{B} \odot \mathbf{A})} \right) \tag{S.17}$$

Figure S.3: *Cascaded four-layer display using a two-frame factorization. In this simulated example, the "drift" image was spatially superresolved by a factor of* 16 *using a stack of four light-attenuating layers, each shifted by* $1/4$ *of a pixel, along each axis. The target image, the depiction with a single (low-resolution) display layer, and the reconstruction using a cascaded four-layer display are shown from left to right. Notice the significant upsampling achieved by the cascaded four-layer display. (Motorsport image courtesy Aurélien Vialatte.)*

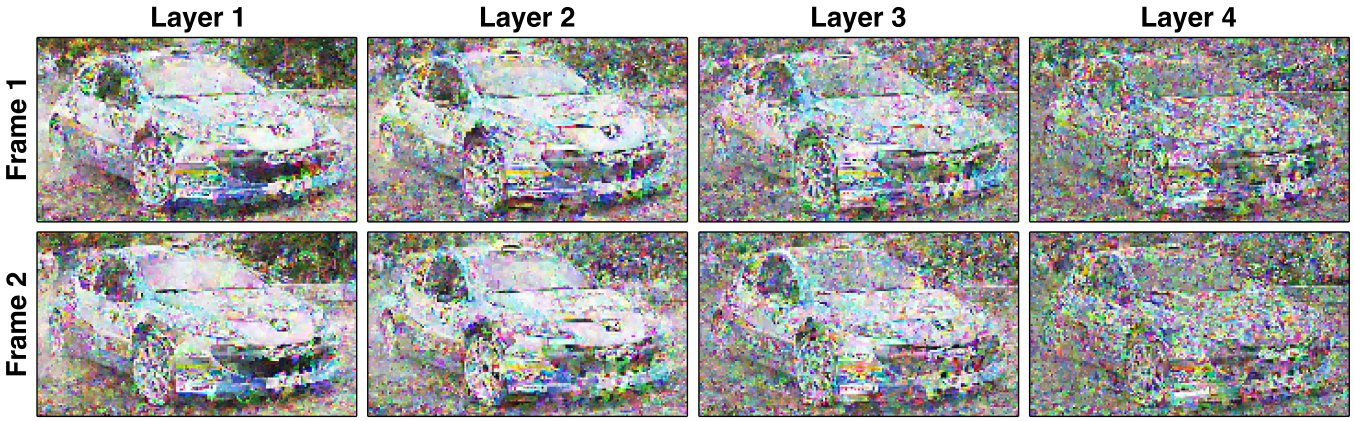

Figure S.4: *Factorized layers and frames for the example in Figure S.3.*

For completeness, we reiterate from that work that, in these expressions, $\odot$ expresses the *Khatri-Rao product*:

$$\mathbf{X} \odot \mathbf{Y} = [\mathbf{x}_1 \star \mathbf{y}_1,\ \mathbf{x}_2 \star \mathbf{y}_2,\ \cdots,\ \mathbf{x}_K \star \mathbf{y}_K]. \tag{S.18}$$

Following Kolda and Bader [2009], $\mathbf{X}_{(n)}$ is the *unfolding* of tensor $\mathcal{X}$, which arranges the mode-n fibers of $\mathcal{X}$ into sequential matrix columns. Generalization to higher factorization orders (i.e., greater numbers of layers) is provided in that reference, as well as Wetzstein et al. [2012], and Ho [2008].

Simulated factorization results are shown in Figure S.3, with the individual layers and frames presented in Figure S.4. Note that we generalize the lateral offset to maximize the superresolution capability: by progressively shifting each layer by $1/4$ of a pixel—creating 16 times as many subpixel fragments as pixels on a single layer. Given this aggressive superresolution goal, we report that single-frame (i.e., order-4, rank-1) factorizations do not accurately reproduce a target high-resolution image. However, we find that two-frame (i.e., order-4, rank-2) factorizations achieve such high superresolution factors, as demonstrated by the fidelity of the inset regions in Figure S.3.

In summary, we have provided a generalized framework for cascaded displays that encompasses arbitrary numbers of offset pixel layers and numbers of time-multiplexed frames; however, we emphasize that cascaded dual-layer displays remain our central focus: providing a means to quadruple spatial resolution with practical display architectures supported by real-time factorization methods (e.g., the cascaded LCD screen and LCoS projector prototypes).

D Details on CFAs for Cascaded Displays

LCD panels primarily achieve color display by the addition of a color filter array, comprising a periodic array of spectral bandpass filters. Typically, three neighboring columns of individually-addressible subpixels, illuminated by a white backlight, are separately filtered into red, green, and blue wavelength ranges, together representing a single full-color pixel column. At sufficient viewing distances, spatial multiplexing of color channels becomes imperceptible. We observe that cascaded dual-layer LCDs can still double the *vertical* resolution when vertically-aligned CFAs are present on each layer. However, increasing the horizontal resolution proves problematic without modifying the CFA structure. We propose two such modifications: the use of multiple color filters per pixel (on the top-most layer) and the use of cyan-yellow-magenta CFAs; use of both results in cascaded dual-layer LCDs that appear as an single LCD with twice the number of color subpixels along each axis.

Each subpixel fragment may depict a different color if it has an independent color filter. As a result, we propose constructing cascaded dual-layer LCDs using *monochromatic* panels (i.e., those free of any color filter arrays). As shown in the Figure 1 in the paper, offsetting such displays by half a pixel, both horizontally and vertically, creates four times as many subpixel fragments as pixels on a single layer. To create a spatially-multiplexed color display, a custom CFA is fabricated that has one color filter per subpixel fragment. This can be achieved by fabricating one panel with a CFA with half the pitch as a conventional panel, such that two vertically-aligned color filters are present at each pixel in the outermost display panel.

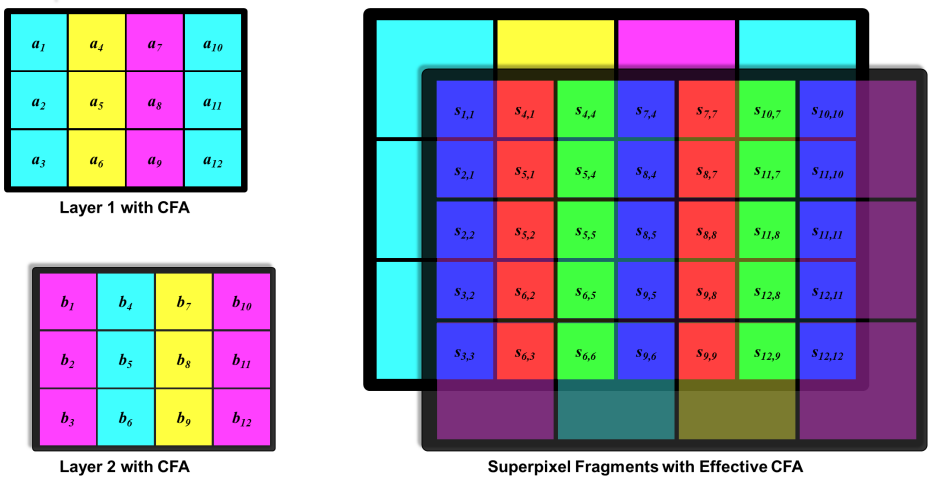

Figure S.5: *Creation of subpixel fragments by dual-layer cascaded displays with cyan-yellow-magenta color filter arrays (CFAs). (Top Left) Similar to Figure 1 in the paper, the rear layer of a cascaded dual-layer display is fabricating similar to existing displays: a fixed CFA is manufactured, such that a single filter acts on each column of pixels. Unlike conventional displays, traditional red-greed-blue filters are replaced with cyan-yellow-magenta triplets. (Bottom Left) A second light-absorbing display is placed in direct contact with the rear display layer, which an identical CFA. (Right) Once again, the geometric overlap of offset pixel layers creates an array of subpixel fragments. The spectral overlap of the color filters creates an effective CFA that appears as a traditional red-greed-blue filter pattern with twice the pitch as the underlying CFAs. Note that the effective CFA could also be achieved simply by manufacturing one of the layers using a red-greed-blue CFA with twice the normal pitch, with no CFA placed in the other layer.*

Fabricating higher-frequency CFAs may require significant engineering investment. As an alternative, we propose stacking two LCD panels with identical color filter arrays (see Figure S.5). However, unlike conventional LCDs with red, green, and blue filters, we propose substituting materials capable of transmitting cyan, yellow, and magenta wavelength ranges. The superposition of two dissimilar filters synthesizes red (i.e., combinations of magenta and yellow), green (i.e., combinations of cyan and yellow), and blue (i.e., combinations of cyan and magenta) colors.

Consider a pair of LCDs with periodic columns of cyan, yellow, and magenta filters, beginning with a cyan column on the left-hand side. We propose positioning the second panel with an offset of one-and-a-half pixels to the right and half a pixel up or down (see Figure S.5). Such a configuration appears with twice as many subpixel fragments along each dimension, covered by what appears to be a conventional red-green-blue CFA with twice the pitch of the CFA in each layer.

This idea can be extended to other sub-pixel layouts and color filters, such as a 2×2-grid of cyan, yellow, magenta, and white. When offset by a quarter pixel in each dimension, we quadruple resolution, but now have apparent cyan, yellow, magenta, red, green, blue, and white sub-pixels. We emphasize that our description of multi-layer cyan-yellow-magenta CFAs is not all-encompassing, and is offered as an illustrative example. As with the 2×2-grid, more general CFA patterns and filter bandpass spectra can be used with the basic principle: overlapped CFAs can synthesize arbitrary target CFAs that modulate individual subpixel fragments, while utilizing existing display manufacturing processes that create a single color filter per pixel, per display layer.

We emphasize that emerging high-speed LCDs may eliminate the need for CFAs, instead using field-sequential color (FSC), in which monochromatic panels sequentially display each color channel, while the backlight color is altered.

E Extended Analysis

E.1 Simulated Quality Comparison on a Dataset of Natural Images

Table S.1 and Table S.2 analyze the quality of a set of superresolved images using three alternative display superresolution techniques (in terms of PSNR and SSIM): additive superresolution, optical pixel sharing, and our cascaded displays. The conclusion from the paper holds here as well. Single-frame cascaded displays achieve a better quality than two-frame additive superresolution displays, both in terms of PSNR and SSIM. Cascaded displays achieve roughly the quality of a two-frame OPS display: the average PSNR of single-frame cascaded displays is slightly less than for the jointly optimized OPS (our improvement to the original OPS paper), but our average single-frame SSIM is slightly better than jointly optimized OPS; overall we consider the quality roughly equal. Cascaded displays with two or more frames outperform all other methods by significant margins.

	conventional	additive (two frames)	additive (four frames)	OPS (two frames, edge-optimal)	OPS (two frames, edge&smooth-optimal)	cascaded (one frame)	cascaded (two frames)	cascaded (three frames)	cascaded (four frames)	OPS (two frames, edge-optimal, **per-image**)	OPS (two frames, edge&smooth **per image**)
Agama	29.34	35.64	46.09	33.01	36.28	35.70	49.34	69.59	90.67	33.10	36.28
Baba in Nepal	26.11	31.84	43.19	28.40	30.53	31.86	46.54	72.29	94.24	28.43	30.53
Bamberg	23.81	29.80	35.36	27.49	31.20	29.94	43.21	64.70	75.53	27.64	31.20
Bird	27.66	33.65	41.48	30.80	33.81	34.02	46.70	66.60	79.41	30.85	33.81
Drift	28.06	36.44	44.98	32.99	36.51	36.61	51.28	72.51	87.37	33.04	36.51
Farmer	27.81	33.37	44.20	30.79	33.48	33.44	48.16	71.18	90.60	30.79	33.48
Fire	29.25	35.12	41.12	33.96	37.91	36.35	49.74	65.19	74.29	33.97	37.91
Fly	24.66	29.12	34.70	30.46	34.00	29.46	39.71	54.10	68.08	30.49	34.00
Gypful	31.25	37.16	44.74	38.24	44.25	37.73	51.47	68.02	80.51	38.35	44.25
Laser	31.05	35.59	45.98	34.53	36.86	36.49	53.34	79.92	94.54	34.53	36.86
Libelle	28.09	33.77	41.42	32.74	36.49	34.02	45.57	63.69	75.56	32.75	36.49
Painted Ladies	27.19	34.87	43.08	31.37	34.59	34.82	50.97	70.35	84.88	31.39	34.59
Porsche	25.64	31.58	35.81	30.59	34.47	31.95	45.38	64.03	78.30	30.82	34.47
Suzuka	27.42	33.12	38.52	34.71	38.96	33.78	45.51	62.24	74.97	34.71	38.96
Townhall	24.81	30.12	36.41	30.17	33.99	30.07	42.51	66.00	85.15	30.18	33.99
Trapp	28.18	34.89	41.78	35.95	39.18	35.72	47.84	69.62	83.74	36.00	39.18
Vespula	27.38	32.51	42.22	30.49	33.00	32.80	45.18	64.76	79.80	30.57	33.00
Glarus Cow	25.04	29.40	37.29	27.78	30.20	29.68	42.93	66.88	80.96	27.85	30.20
Melinaea	32.00	38.90	47.43	39.33	44.75	39.50	53.98	70.18	83.50	39.38	44.75
Mototaxis	29.37	35.88	45.17	33.16	36.49	36.26	49.63	70.09	84.20	33.17	36.49
AVG	**27.71**	**33.64**	**41.55**	**32.35**	**35.85**	**34.01**	**47.45**	**67.60**	**82.32**	**32.40**	**35.85**

Table S.1: *Peak signal-to-noise (PSNR) in [dB] for a set of natural images. Three alternatives are compared: additive superresolution displays using either two or four frames, optical pixel sharing (OPS) using two frames, and cascaded displays using one, two, three and four frames. We emphasize that additive superresolution uses a single display layer, whereas OPS and cascaded displays employ two display layers. For OPS we have included two versions, one where we have optimized its edge-theshold and used* $1/\varepsilon = 8$ *for smoothing as described in the paper. We have also included a second version where we have jointly optimized for both, the edge-threshold and the smoothing parameter* $1/\varepsilon$*, which was not described in the original paper. For the optimization of the optimal parameters for this image set, we have used the average PSNR in the last row of this table as the objective function. For the table on the right (in grey) we have optimized the OPS parameters per image for the best achievable quality.*

E.2 Spatial Superresolution: Slanted Edge MTF Measurements

We also compare MTFs that are computed using the slanted edge method [Burns 2000] in Figure S.6. In this case, the MTF is estimated from the profile of the slanted edge. Note how we the slanted edge MTF of our cascaded display matches the MTF of the target image. We also observe that OPS reproduces the slanted edge very well, since there is enough pixel intensity in the bright regions that it can redistribute to the edge. Please note that this breaks down for natural images (as shown in the section above and for more challenging frequency charts, i.e., the MTF via chirp in the paper).

	conventional	additive (two frames)	additive (four frames)	OPS (two frames, edge-optimal)	OPS (two frames, edge&smooth-optimal)	cascaded (one frame)	cascaded (two frames)	cascaded (three frames)	cascaded (four frames)	OPS (two frames, edge-optimal, **per-image**)	OPS (two frames, edge&smooth **per image**)
Agama	2.6819	2.9338	2.9948	2.8110	2.8898	2.9347	2.9957	2.9998	3.0000	2.828	2.890
Baba in Nepal	2.3541	2.8304	2.9867	2.5766	2.6989	2.8309	2.9941	3.0000	3.0000	2.578	2.699
Bamberg	2.4238	2.8530	2.9664	2.6849	2.8079	2.8582	2.9939	2.9999	3.0000	2.685	2.808
Bird	2.5995	2.8977	2.9833	2.7635	2.8512	2.9076	2.9948	2.9999	3.0000	2.765	2.851
Drift	2.6366	2.9347	2.9952	2.7703	2.8567	2.9383	2.9979	3.0000	3.0000	2.788	2.857
Farmer	2.4702	2.8611	2.9911	2.6746	2.7938	2.8621	2.9945	2.9999	3.0000	2.685	2.794
Fire	2.7300	2.9278	2.9822	2.8539	2.9108	2.9455	2.9975	2.9999	3.0000	2.861	2.911
Fly	2.5230	2.8480	2.9653	2.8059	2.8885	2.8591	2.9909	2.9997	3.0000	2.808	2.889
Gypful	2.8332	2.9515	2.9916	2.9227	2.9729	2.9563	2.9976	2.9998	3.0000	2.942	2.973
Laser	2.3728	2.7674	2.9792	2.6453	2.7673	2.7936	2.9956	3.0000	3.0000	2.651	2.767
Libelle	2.7461	2.9319	2.9879	2.8552	2.9125	2.9373	2.9957	2.9998	3.0000	2.868	2.913
Painted Ladies	2.5505	2.9046	2.9860	2.7454	2.8435	2.9050	2.9973	2.9999	3.0000	2.754	2.844
Porsche	2.5691	2.8784	2.9672	2.7760	2.8664	2.8858	2.9959	2.9999	3.0000	2.779	2.866
Suzuka	2.6407	2.9244	2.9847	2.7885	2.8694	2.9339	2.9965	2.9999	3.0000	2.808	2.869
Townhall	2.5729	2.8691	2.9712	2.8250	2.9047	2.8688	2.9932	2.9998	3.0000	2.831	2.905
Trapp	2.8018	2.9700	2.9950	2.9100	2.9472	2.9753	2.9986	3.0000	3.0000	2.915	2.947
Vespula	2.4246	2.8383	2.9818	2.6663	2.7743	2.8501	2.9928	2.9999	3.0000	2.666	2.774
Glarus Cow	2.1670	2.7186	2.9634	2.4766	2.6263	2.7331	2.9893	2.9999	3.0000	2.477	2.626
Melinaea	2.8529	2.9548	2.9949	2.9171	2.9594	2.9569	2.9978	2.9999	3.0000	2.932	2.959
Mototaxis	2.6415	2.9205	2.9917	2.7765	2.8611	2.9245	2.9968	3.0000	3.0000	2.788	2.861
AVG	**2.5796**	**2.8858**	**2.9830**	**2.7623**	**2.8501**	**2.8928**	**2.9953**	**2.9999**	**3.0000**	**2.771**	**2.850**

Table S.2: *Structural similarity index (SSIM [Wang et al. 2004]) as a sum over all color channels for a set of natural images. Three alternatives are compared: additive superresolution displays using either two or four frames, optical pixel sharing (OPS) using two frames, and cascaded displays using one, two, three and four frames. We emphasize that additive superresolution uses a single display layer, whereas OPS and cascaded displays employ two display layers. For OPS, we have included two versions, one where we have optimized its edge-theshold and used* $1/\varepsilon = 8$ *for smoothing as described in the paper. We have also included a second version where we have jointly optimized for both the edge-threshold and the smoothing parameter* $1/\varepsilon$*, which was not described in the original paper. For the optimization of the optimal parameters for this image set, we have used the average SSIM in the last row of this table as the objective function. For the table on the right (in grey) we have optimized the OPS parameters per image for the best achievable quality.*

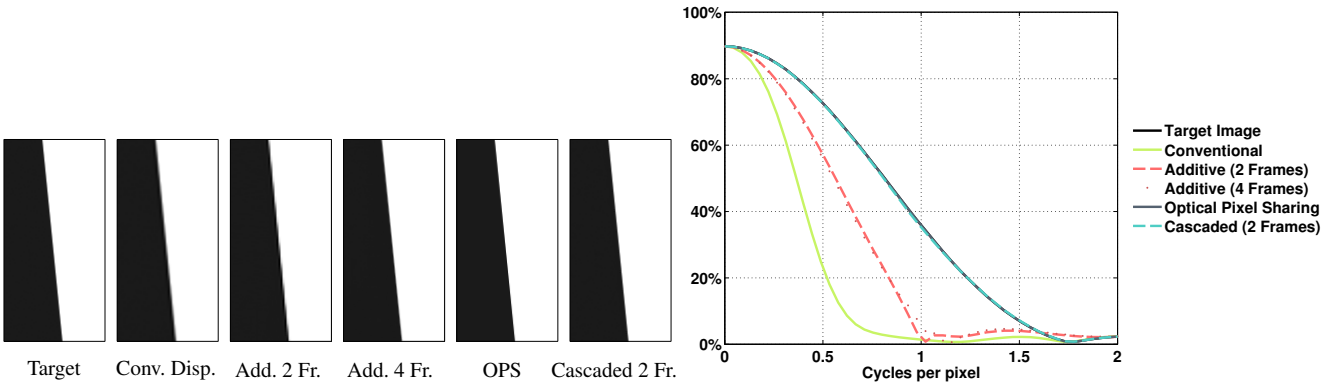

Figure S.6: *Slanted edge comparison between target image, conventional display, additive displays with 2 and 4 frames, OPS, and cascaded displays (rank-2). On the right, we show the slanted edge MTF measurement for the different methods.*

E.3 HDR Analysis of Cascaded Displays

Similar to other dual-modulations displays reviewed in Section 2.2, cascaded displays also increase the dynamic range. Figure S.7 presents the appearance of a linear ramp using a a pair of 8-bit cascaded displays. As observed throughout this section, reconstruction artifacts due to compression are nearly eliminated by adopting two-frame factorizations.

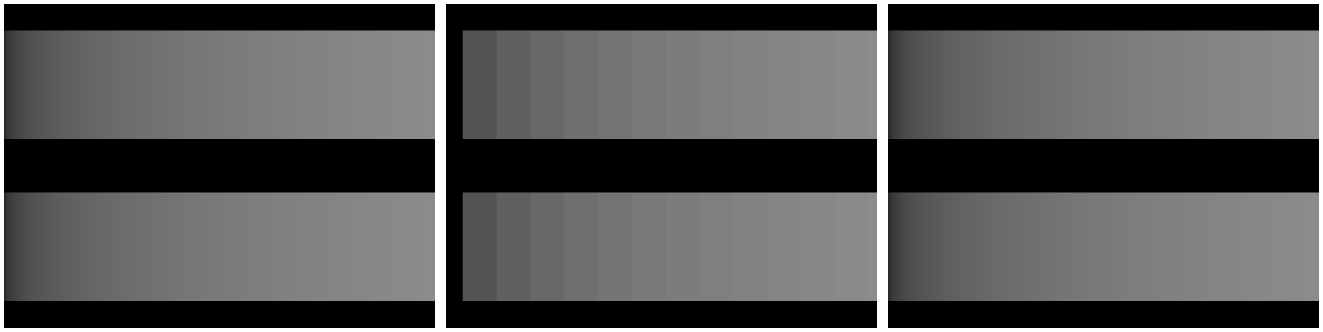

Figure S.7: *High dynamic range (HDR) applications of cascaded displays. A target ramp (left) is presented with a single* 8*-bit display (middle) and a cascaded display using two* 8*-bit layers (right).*

E.4 Analysis of Temporal Superresolution on a Natural Movie

In the following, we compare the quality of our temporal superresolution vs. the at the original lower frame rate. To this end, we compute the PSNR and SSIM between the target video at superresolved frame rates and our method and the normal-framerate (i.e., low-framerate) video (called "naive" in the figure.).

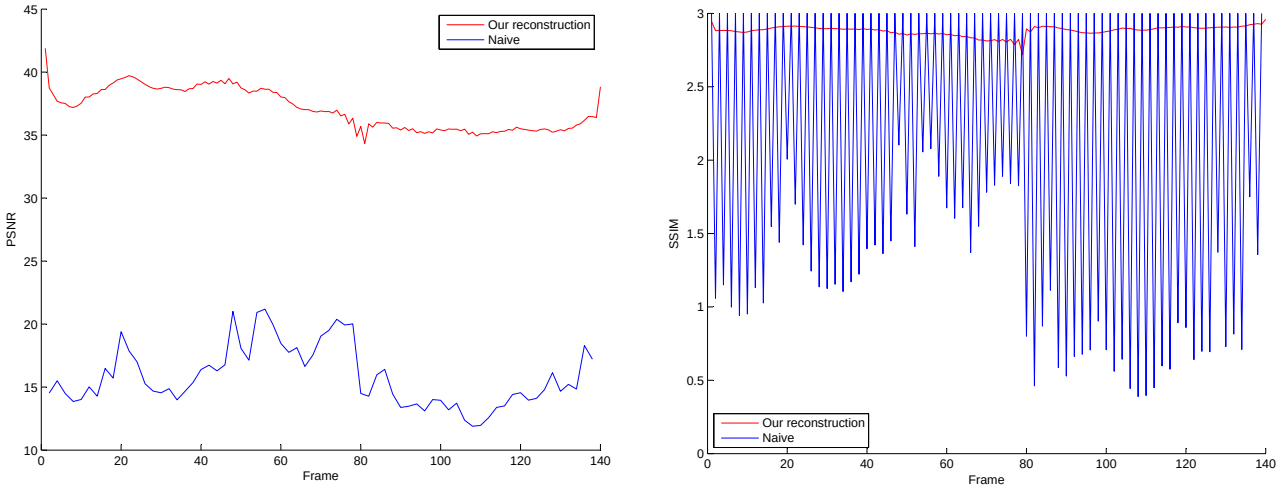

Figure S.8: *PSNR and SSIM over movie frames ("drift sequence", see video supplemental).*

Supplementary References

BLONDEL, V., HO, N.-D., AND VAN DOOREN, P. 2008. Weighted nonnegative matrix factorization and face feature extraction. In *Image and Vision Computing*, 1–17.

BURNS, P. D. 2000. Slanted-edge mtf for digital camera and scanner analysis. In *Proc. IS&T 2000 PICS Conference*.

HO, N.-D. 2008. *Nonnegative Matrix Factorization Algorithms and Applications*. PhD thesis, Université catholique de Louvain.

KOLDA, T. G., AND BADER, B. W. 2009. Tensor decompositions and applications. *SIAM Review 51*, 3, 455–500.

LEE, D. D., AND SEUNG, H. S. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature 401*, 6755, 788–791.

PAATERO, P., AND TAPPER, U. 1994. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics 5*, 2, 111–126.

WANG, Z., BOVIK, A., SHEIKH, H., AND SIMONCELLI, E. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4, 600–612.

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. 31*, 4, 80:1–80:11.