# Video Killed The Data Store

## Extending the n-Dimensional Display Interface for Full Screen Video

Charles D. Estes
University of North Carolina at Chapel Hill
Brooks Computer Science Building, CB 3175
Chapel Hill, NC 27599-3175 USA
cdestes@cs.unc.edu

Ketan Mayer-Patel
University of North Carolina at Chapel Hill
Brooks Computer Science Building, CB 3175
Chapel Hill, NC 27599-3175 USA
kmp@cs.unc.edu

## ABSTRACT

Prior research introduced the n-Dimensional Display Interface (NDDI) as a new *"narrow waist"* for the display pipeline. In this paper, we extend the NDDI architecture to provide a blending feature. We then utilize that new feature for full screen video playback, leveraging application-level framing to realize a significant data transmission reduction. We then explore new NDDI configurations for effective rate control under highly constrained transmission budgets.

## Categories and Subject Descriptors

C.0 [**General**]: Hardware/software interfaces

## General Terms

Design, Algorithms, Performance, Experimentation

## Keywords

Display interface, framebuffer, scalable display

## 1. INTRODUCTION

Ever increasing display resolutions are pushing the limits of framebuffer-based computer/display interfaces. Progressively scanning a framebuffer at 5K display resolutions at a fixed synchronous rate of 60Hz and 24 bits per pixel, for example, requires 19 Gb/s of raw throughput. These data rates are particularly challenging when connecting to a display wirelessly or using mobile devices with a limited battery life. Clearly, these data rates motivate the use of compression in some way. Choosing a single compression standard for communicating with displays, however, is problematic. Different use cases and types of content will require different types of compression techniques. Settling on a single fixed standard limits future innovation and creates interoperability issues.

Our approach to this challenge is to rethink the computer/display abstraction entirely. By replacing the simple framebuffer abstraction with a more general and flexible abstraction that can be used to bring application-level domain knowledge to bear in how display resources are best marshaled. In effect, we have proposed a display architecture that can be used to realize application-specific display compression. We call this abstraction the n-Dimensional Display Interface (NDDI) [7].


*MM'15,* October 26–30, 2015, Brisbane, Australia.

ACM 978-1-4503-3459-4/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2733373.2806271 .

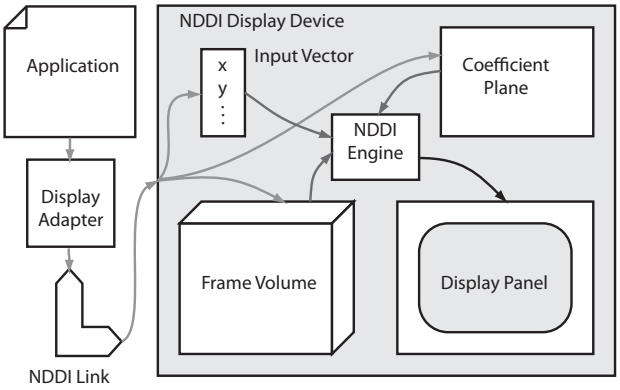


**Figure 1: NDDI concept diagram.**

One way to think about a display is to consider it a sort of *"big data problem"* where massive amounts of graphical data are mapped and reduced to a 2D array of pixels on a display. NDDI is designed to be a mechanism that allows applications to explicitly express the contents of a display as a collection of map-reduce operations on pixel data. It is intended to act as a *"narrow waist"* abstraction that many different kinds of applications will find different ways of employing.

The NDDI Display concept is illustrated in Figure 1. Instead of a simple framebuffer matching the dimensions of the display, NDDI uses a *Frame Volume* that can be configured to any dimensionality. It serves largely as a store of pixel data on the NDDI device that can be mapped to the *Display Panel*. The mapping is controlled by the *Coefficient Plane*, which is a two-dimensional array of *Coefficient Matrices*. The dimension of the Coefficient Plane match that of the physical Display Panel (i.e., there is a Coefficient Matrix for each pixel on the display). Updates to the display are driven by the *Input Vector*. The *NDDI Engine* houses digital logic that performs calculations necessary to multiply the Input Vector by each pixel's Coefficient Matrix to produce an address tuple. The length of the address tuple matches the dimensionality of the Frame Volume and acts to address a specific pixel value stored in the Frame Volume

which is retrieved and placed on the Display Panel. Absent of data dependencies, these calculations can all happen in parallel.

By configuring the size of the Input Vector and dimensionality of the Frame Volume, applications can tailor the display to their needs instead of interacting via a fixed framebuffer. This application-level framing [1] empowers the application to reconfigure the display based on its semantics. For example, a remote kiosk cycling through a set of slides might pre-render its content to the Frame Volume and then slowly switch between advertisements with a single byte written to the appropriate element of the Input Vector. An interactive display with an intricate graphical user interface might configure the Frame Volume as a 4D storage for a variety of content like rendered frames, graphics primitives, icons, fonts, sprites, etc. Compositing could then be performed through appropriate mappings in the Coefficient Plane.

NDDI is not meant as a replacement for a GPU, nor is it a special purpose display optimized for video playback. It is simply a particular type of display that can be connected to a display adapter with a GPU. The GPU can drive the NDDI Display like a framebuffer or in numerous more advanced ways to realize a benefit based on application-level semantics.

Our initial experiments focused on traditional applications with no knowledge of the higher-order functionality provided by NDDI. These experiments simply read out the framebuffer on every refresh cycle and translated the data into NDDI-specific commands to update the memory regions of the NDDI Display. The first significant benefit we demonstrated was to update the NDDI Display only when the contents of the framebuffer changed instead of on a fixed refresh rate of 60 Hz. Subsequent experiments improved these results by using a tiled approach that divided the framebuffer into blocks and updated the Frame Volume on a block-by-block basis, potentially reusing previously transmitted blocks.

The results were very compelling for many different computer applications, save for one. Full screen video playback saw no real benefit beyond updating the NDDI display at 24 fps instead of 60 Hz. Furthermore, the original NDDI Display specification lacked a critical feature: blending. Several use cases require blending to smoothly composite non-rectangular regions. This paper describes a refined NDDI design that adds this critical new feature and uses it in a novel way to revisit rendering of full screen video. The experiments within show a 75% reduction in data transmission cost placing video performance on par with the other computer applications from the prior experiments.

In the remainder of this paper we provide more details on the NDDI in Section 2. In Section 3 we discuss blending extensions needed for advanced compositing. In Section 4 those extensions are used to drive NDDI in a novel video-specific manner. We then extend these ideas in Section 5 to include a multiscale approach that lends itself to constrained bandwidth situations. Related work that inspired these new schemes are described in Section 6 and the paper concludes with a discussion of future work in Section 7.

# 2. N-DIMENSIONAL DISPLAY INTERFACE

NDDI is a display interface that uses a more complex abstraction to replace the framebuffer. Defining this abstraction required balancing tradeoffs between transmission benefit and flexibility. Our design concept for the new abstraction was driven by a set of guiding principles.

**Framebuffer Compatible** - Should be fully backward compatible with applications that use a simple framebuffer.

**Data-Driven** - Should be stateless, updating deterministically based on data alone.

**Progressive Benefit** - Applications can leverage application-level framing for stronger benefit.

**Highly Parallel** - Operations should remain independent allowing for massive scalability.

**Asynchronous** - Should not require a synchronous signal to drive updates.

## 2.1 NDDI Components

As introduced earlier, the NDDI Frame Volume ($\mathbf{V}$) stores 3-channel pixel values transmitted to the display by an application. Once in the Frame Volume, these pixel values can be used in a number of ways, depending on the needs of the driving application. A key innovation of the Frame Volume is that it can be dynamically configured to any $n$-dimension pixel space ($\mathbb{P}^n$), allowing the pixels to be addressed with a $n$-tuple matching that dimensionality. This allows applications to explicitly express the contents of a display as a "mapping" of these pixels. With the data now on the display, these mapping operations can scale with the display instead of relying on rendering and transmission components to scale in lockstep.

Specifically, the mapping operations performed by NDDI are driven by the Input Vector ($\mathbf{i}$) and controlled by the Coefficient Plane ($\mathbf{C}$). The Input Vector is a one dimensional vector with a length ($m$) that is configureable such that $m \geq 2$. The first two elements $i_x$ and $i_y$ are reserved and logically are set to the $x$ and $y$ coordinates for the pixel being mapped. The remaining values are to be updated by the application on a frame-by-frame basis. The Coefficient Plane is a two-dimensional array matching the physical dimensions of the Display Panel ($w \times h$). The Coefficient Plane holds Coefficient Matrices ($\mathbf{M}$). A Coefficient Matrix is a matrix of Coefficients ($c$) where the size of the matrix is coordinated with the length of the Input Vector and the dimensionality of the Frame Volume ($m \times n$). The purpose of each Coefficient Matrix at location $(x, y)$ in the Coefficient Plane is to perform a mapping that chooses a pixel from the Frame Volume to be composited (i.e., reduced) on to the Display Panel at location $(x, y)$. The mapping is done by performing matrix multiplication of the Input Vector by the Coefficient Matrix, producing an $n$-tuple that uniquely addresses a pixel value from the framebuffer.

## 2.2 Pixel Mapping

The pixel mapping operations are deterministic, consisting of highly parallel matrix multiplication. The equations below provide formal definitions. $p_{x,y}$ is the value from pixel space being mapped at location $x, y$ on the Display Panel (2). The Frame Volume ($\mathbf{V}$) is configured with $n$ fixed dimensions (an $n$th-order tensor), a subset of $n$ pixel space (3). The Input Vector ($\mathbf{i}$) is an integer vector of length $m$ (4). Each Coefficient Matrix ($\mathbf{M}$) is an $m \times n$ matrix of coefficients ($c$) (6). Coefficients are integers (5). The Coefficient Plane ($\mathbf{C}$) is a two dimensional array from coefficient matrix space.

$$\mathbb{P} = \{(r, g, b) | 0 \leq r, g, b < 2^8\} \quad (1)$$

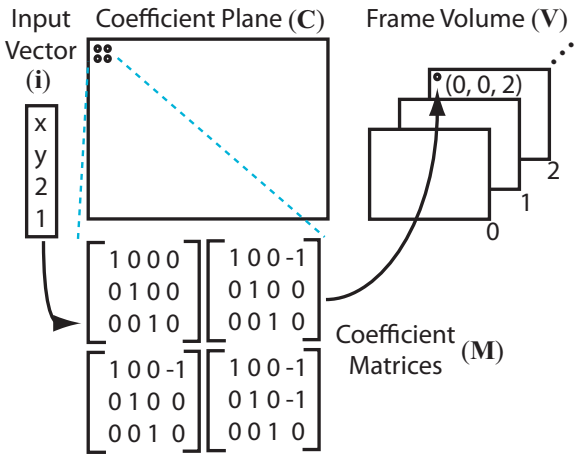


**Figure 2: Example NDDI Configuration for simple video player with 4x scaling.**

$$p_{x,y} \in \mathbb{P}, x < w, y < h \tag{2}$$

$$\mathbf{V} = V_{i_1 i_2 \ldots i_n}, v \in \mathbb{P} \tag{3}$$

$$\mathbf{i} \in \mathbb{Z}^m \tag{4}$$

$$c \in \mathbb{Z} \tag{5}$$

$$\mathbf{M} = \begin{bmatrix} c_{11} & \cdots & c_{1m} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nm} \end{bmatrix} \tag{6}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{M_{11}} & \cdots & \mathbf{M_{1w}} \\ \vdots & \ddots & \vdots \\ \mathbf{M_{h1}} & \cdots & \mathbf{M_{hw}} \end{bmatrix}, \mathbf{C}(x,y) = \mathbf{M_{xy}} \tag{7}$$

The Lookup operation takes an $n$-tuple as an address and retrieves the pixel from that location of the Frame Volume (8). The Map operation accepts a 2-tuple $(x, y)$ and uses the Coefficient Matrix at that location in the Coefficient Plane to multiply by the Input Vector (9). The first two values of the Input Vector will be set to $x$ and $y$ and the remaining values are set by the application.

$$p = \text{Lookup}(\mathbf{V}, (a_1, ..., a_n))) \tag{8}$$

$$(a_1, ..., a_n) = \text{Map}(x, y) = \mathbf{C}(x, y)\mathbf{i} \tag{9}$$

The complete calculation for the pixel $p_{x,y}$ uses the Map operation to produce the address tuple used in the Lookup (10).

$$p_{x,y} = \text{Lookup}(\mathbf{V}, \text{Map}(x, y)) \tag{10}$$

As an example, Figure 2 shows a possible configuration for a scaled video player application. The Input Vector is configured with length four. The Frame Volume is configured with three dimensions and each frame of video is rendered into an $x$-$y$ plane of the Frame Volume, treating the $z$ dimension like a circular buffer. The Coefficient Matrices are therefore defined to have four columns and three rows. The application chooses coefficients to specify an affine transformation so that the contents of each $x$-$y$ plane are effectively scaled by four. The third value in the Input Vector is then used to choose which $x$-$y$ plane is active (i.e., advancing through the circular buffer) and the fourth value is fixed at one as part of the affine transformation.

The following shows the specific calculation for the pixel at $(1, 1)$ of the display using the values from Figure 2. This is the fourth pixel in this group of pixels that are scaled from a single pixel.

$$\text{Map}(1,1) = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \tag{11}$$

$$p_{1,1} = \text{Lookup}(\mathbf{V}, (0, 0, 2)) \tag{12}$$

Map-reduce as a general paradigm is a natural fit for our goal of providing a scalable display interface abstraction. NDDI uses map-reduce at multiple levels in order to arrive at a result. At the coarsest level, NDDI uses the Coefficient Plane to map a pixel value to each Display Panel location. At a finer level, the matrix multiplication of the Input Vector by each Coefficient Matrix can be considered a map-reduce operation producing a tuple that chooses a pixel value from the Frame Volume. Later in the paper, we will refine the NDDI design with another map-reduce layer to provide pixel blending operations that can also be used to support video.

## 2.3 Application Agnostic Experiments

Our prior experiments assumed the application to be unaware of NDDI and aimed to demonstrate a transmission savings without leveraging any application-level semantics. These experiments served to support backward compatibility by monitoring a framebuffer and translating its changes into the appropriate NDDI commands. The first experiment configured NDDI as a simple framebuffer, while the next two experiments used "tilers" that divided the framebuffer into blocks that could be updated and reused as necessary.

**Simple Framebuffer Experiment** - Treats the entire framebuffer as a single block, updating the entire region whenever a pixel changes. The Frame Volume is organized into two dimensions matching the Display Panel.

**Flat Tiled Experiment** - Tiles the framebuffer into blocks and only updates the blocks that change. The Frame Volume is configured to match the Display Panel.

**Cached Tiled Experiment** - In addition to tiling the framebuffer, the Frame Volume is organized into three dimensions forming a very deep stack of 2D tiles (Figure 3) that act as a cache of blocks. A block is easily mapped to the Display Panel by using an efficient Coefficient Plane fill command for that region of Coefficient Matrices.

These initial experiments were driven with pre-recorded computing sessions at 24 fps. The results for the computing sessions are shown in Figure 4. The results are shown as a ratio of the NDDI data sent versus the amount of data that would be required if the entire display was scanned and transmitted every 60 Hz cycle. As a best case, an *Ideal* mode was introduced. It represents the amount of data required to send over just the individually changed pixels, ignoring the cost of addressing that pixel on the display.

In the Simple Framebuffer experiment, updating the NDDI Display at 24 fps instead of 60Hz provides an immediate

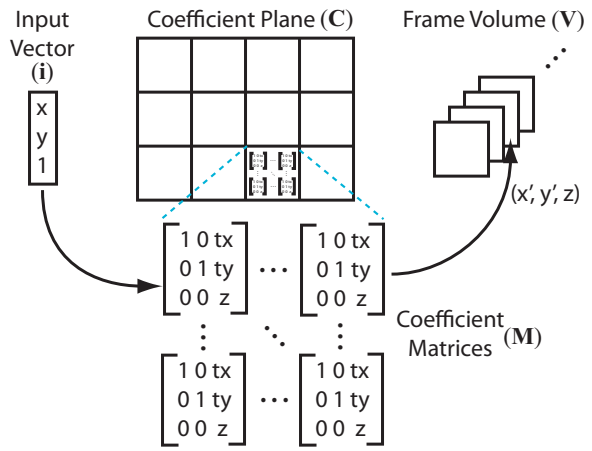


**Figure 3: Cached Tiler NDDI configuration.**

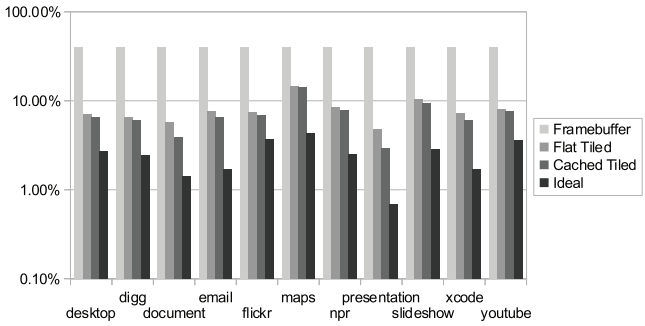


**Figure 4: Experiment results for the recorded computing sessions (logarithmic scale).**

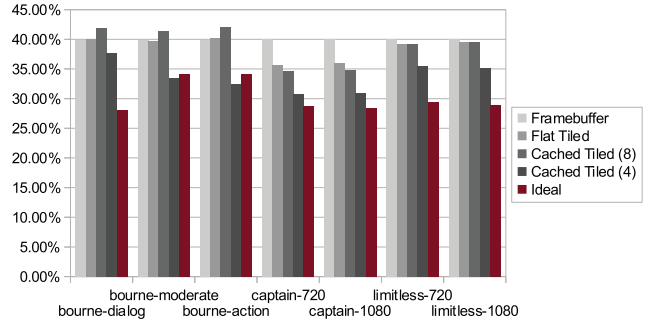


**Figure 5: Experiment results for full screen video playback.**

benefit by reducing how often data was sent. Furthermore, any time the display was unchanged, no data was sent at all. The Flat Tiled experiment did significantly better for scenarios where the screen was mostly static. The Cached Tiled experiment did marginally better than Flat Tiled. It benefited from efficient fill operations where a single block could be used for several different regions of the display. Additionally scenarios that re-used content such as switching between windows showed a significant benefit.

The results for full screen video playback (Figure 5) were very weak by comparison. In an attempt to realize stronger gains, the Cached Tiler was augmented with a lossy mode where only a certain number of significant bits per pixel channel were used when determining cache matches. Specifically, Cached(8) is lossless (i.e., the match must be exact) and Cached(4) is lossy using only the first 4 bits per channel to match blocks. Using Cached(4) a modest gain was realized while maintaining a reasonable PSNR ranging from 36.71 to 48.73. However, visual artifacts from re-using near matches affected perceived quality.

# 3. BLENDING

In order to support non-trivial uses cases that require compositing on the display, NDDI must provide a means to blend pixels. The original NDDI design does not do so. To address this shortcoming, we refined the NDDI design while staying true to our guiding principles. Three different approaches were considered.

## 3.1 Temporal Blending

The first approach requires no modifications to the original NDDI feature set. Temporal Blending exploits image persistence associated with the physical operation of the Display Panel. The driving application renders the planes to be blended into separate planes within the Frame Volume. Pixel blending is then achieved by driving the Input Vector to rapidly switch between those planes using a duty cycle tuned to the desired blending percentage.

Disadvantageously, Temporal Blending is tightly coupled to the technology used in the Display Panel. Different display panel technologies may have vastly different levels of image persistence and may support different refresh rates. Furthermore, it is difficult to blend only sub-regions of the display. Lastly, Temporal Blending is not in keeping with our guiding principle which encourages NDDI to be asynchronous. Driving the Input Vector so precisely and synchronously puts heavy requirements on the quality of service of the connection to the NDDI Display.

## 3.2 Frame Volume Blending

Frame Volume Blending is perhaps the most intuitive approach. With Frame Volume Blending, pixel values are extended to support an alpha channel. The alpha channel is used with a new NDDI command that alpha blends pixel values from one area of the Frame Volume onto another target area. This blending operation might be implemented by a dedicated digital logic block. The application must direct these operations in order to achieve the desired blended result.

Frame Volume Blending suffers from a number of disadvantages. First, the capabilities of any blender-specific hardware will need to scale with the display in order to blend display-sized or larger areas within the Frame Volume. Sec-

ond, Frame Volume Blending upsets the ratio of Frame Volume reads to writes. As originally conceived without Frame Volume Blending, large Frame Volume sizes can be supported with an appropriate memory cache hierarchy. Frame Volume Blending now requires more emphasis on Frame Volume write performance and works against caching. In addition to these engineering disadvantages, Frame Volume Blending like Temporal Blending is not in keeping with our guiding principles. Frame Volume Blending requires the application to send over the components of a blended area and to use multiple NDDI commands to composite the result in stages. This uses NDDI as a sort of state machine rather than being entirely data-driven and highly parallel.

## 3.3 Coefficient Plane Blending

While less obvious, the third approach proved to be the most compatible with our guiding principles while remaining highly flexible. Coefficient Plane Blending extends the NDDI architecture to now provide 64 separate Coefficient Planes and augments every Coefficient Matrix with an associated *Scaler* (13). A Scaler ($\mathbf{s}$) is a 3-tuple representing independent scalers for each color channel (15). Each of the 64 Coefficient Planes still matches the physical dimensions of the display panel.

With this extended architecture, to calculate a pixel value for the Display Panel, the Coefficient Matrices at position $(x, y)$ in each of the 64 Coefficient Planes will map to specific pixel values from the Frame Volume. These 64 pixel values are reduced to a single value by multiplying each by their associated Scaler, accumulating across all 64 planes and dividing by the maximum Scaler ($\mathbf{s}_\gamma$). This is similar to alpha blending, but with a configurable maximum scaler (16) that in our implementation must be a power of two instead of a fixed maximum of 256. This provides a level of flexibility while constraining the maximum to ensure that the division can be implemented efficiently as a shift operation.

$$\mathbf{C_k} = \begin{bmatrix} \mathbf{M_{11}} & \cdots & \mathbf{M_{1w}} \\ \vdots & \ddots & \vdots \\ \mathbf{M_{h1}} & \cdots & \mathbf{M_{hw}} \end{bmatrix}, 0 \leq k < 64, \mathbf{C_k}(x, y) = \mathbf{M_{xy}} \tag{13}$$

$$\mathbf{S_k} = \begin{bmatrix} \mathbf{s_{11}} & \cdots & \mathbf{s_{1w}} \\ \vdots & \ddots & \vdots \\ \mathbf{s_{h1}} & \cdots & \mathbf{s_{hw}} \end{bmatrix}, 0 \leq k < 64, \mathbf{S_k}(x, y) = s_{xy} \tag{14}$$

$$\mathbf{s} = \{(r, g, b) | -2^{15} < r, g, b < 2^{15}\} \tag{15}$$

$$\mathbf{s}_\gamma = \{(2^\gamma, 2^\gamma, 2^\gamma) | 0 \leq \gamma < 15\} \tag{16}$$

Unlike alpha blending, the NDDI scalers are both signed and multi-channel allowing for both positive and negative blending with different values for each color channel. Furthermore, the retrieved pixel values can be treated as unsigned 8-bit values or as signed 7-bit values allowing subtractive masks. The Map operation as defined before is modified slightly to take an additional plane argument (17). The Blend operation simply sums the mapped values multiplied (dot product) by the scalers and then shifts by $\gamma$ (18).

$$(a_1, ..., a_n) = \text{Map}(x, y, k) = \mathbf{C_k}(x, y)\mathbf{i} \tag{17}$$

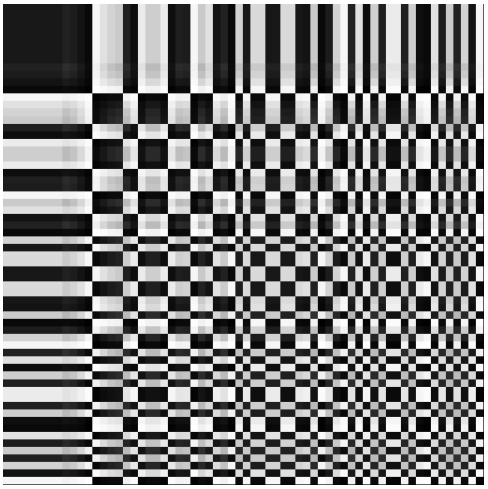

**Figure 6: Rendered DCT basis functions.**

$$(a_1, ..., a_n) = \text{Blend}(x, y) = \frac{\sum_{k=0}^{63} \text{Map}(x, y, k) \cdot \mathbf{S_k}(x, y)}{2^\gamma} \tag{18}$$

While appearing complex, Coefficient Plane Blending is in keeping with the guiding principles – in particular allowing an application to use it in a simple way or to bring to bear application-level semantics to employ it in a more sophisticated manner as is shown in the video use case described in Section 4.

From an engineering perspective, Coefficient Plane Blending does greatly increase the amount of memory required for the Coefficient Planes. That memory, however, can be localized within the per-pixel digital logic. Furthermore, the blending operation itself is just another map-reduce step where the mapping operation performs the scaling and the reduction operation is the accumulate and shift.

# 4. VIDEO

The refined NDDI architecture to support blending can be used to benefit video content as well. Prior experiments showed lackluster performance on video playback in terms of transmission savings. With *Coefficient Plane Blending* in place, we revisit video playback as a task of blending pre-rendered DCT basis functions (Figure 6) [13]. This new experiment introduces our first scheme for configuring the NDDI Display to benefit a application-specific content model.

We implemented a new tiler called the DCT Tiler. The NDDI display was configured as shown in Figure 7. The Frame Volume is configured to be $8 \times 8 \times 64$ so that it can hold 63 pre-rendered basis functions and a level shift plane as described below. The Input Vector has a length of three with the third value fixed at one. The Coefficient Matrices all have simple affine transforms that map each block of a Coefficient Plane back to an $8 \times 8$ area of the Frame Volume with the $z$ component designating a specific plane (i.e., a particular basis function).

For the initial Frame Volume setup, the 63 pre-rendered basis functions were generated as grayscale renderings of a maximum coefficient of 256 clamped to the range $[-127, 127]$. They are arranged in "zig-zag" order which is important

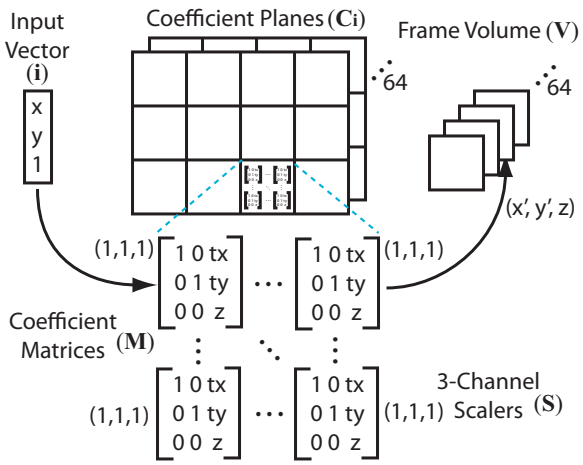


**Figure 7: DCT Tiler NDDI configuration.**

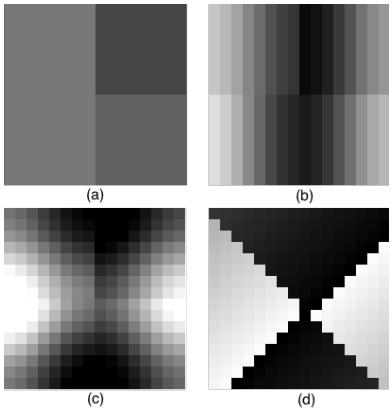


**Figure 8: Example blending of weighted basis functions for four $8 \times 8$ blocks. (a) shows the rendering with just the DC coefficients for the four blocks. (b) blends the first AC coefficient for each block. (c) blends the second AC coefficient. Finally (d) shows the results of all 63 weighted coefficients blended.**

for efficiently updating Scalers associated with a particular block. The final 64th plane is not a pre-rendered basis function, but rather a medium gray plane. This is required because the basis functions are signed and centered at zero. The resulting image must be level shifted to unsigned pixels centered at medium gray.

With the Input Vector, Coefficient Matrices, and Frame Volume initialized, only the Scalers are left uninitialized. The Scalers represent the DCT coefficients. Essentially, the DCT Tiler acts as a simple intra-frame encoder using $8 \times 8$ blocks. For each frame, the video is tiled into blocks. Those blocks are transformed to the frequency domain via a DCT. Then the values are quantized with a configurable quality factor.[1] The values are then de-quantized to produce a set of weights. These are arranged into zig-zag order matching the organization of the basis functions already rendered in the Frame Volume and used to set the Scalers for the block.

[1]The quality factor is an integer multiplied by a simple quantization matrix biased against high-frequency components.[15]

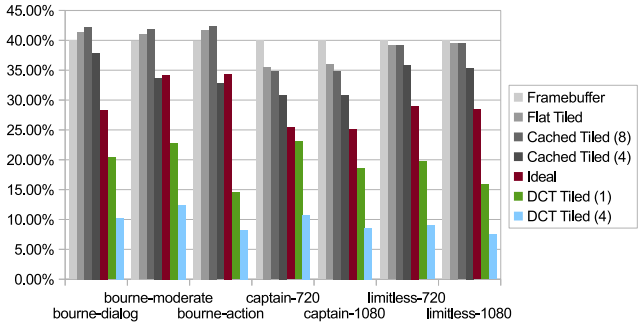


**Figure 9: Revised video results with new DCT tiler.**

| Video | PSNR (1) | PSNR (4) |
|---|---|---|
| bourne-dialog | 42.49 | 39.81 |
| bourne-moderate | 41.35 | 37.09 |
| bourne-action | 43.91 | 42.03 |
| captain-720 | 41.55 | 37.99 |
| captain-1080 | 42.66 | 40.03 |
| limitless-720 | 42.30 | 39.55 |
| limitless-1080 | 43.28 | 41.24 |

**Table 1: DCT Tiled Mode Statistics using a quality factor of 1 (best) and 4 (modest).**

Figure 8 illustrates this process with an example of weighted basis functions being blended over a small $16 \times 16$ region of pixels.

Figure 9 shows the results for video using the new DCT tiler alongside the results from the original experiments. The $y$-axis shows a ratio of the bytes transmitted over the NNDI link for the particular mode compared to the bytes transmitted when the entire framebuffer is sent at 60 Hz. The two additional DCT Tiler runs use a quality factor of 1 producing the highest available quality and 4 which was experimentally determined to reduce the number of surviving coefficients while still producing comparable PSNR scores (see Table 1). The ratio of both of the DCT Tiler runs are far better than Ideal which is a contrived mode that only measures the amount of data associated with pixels that are changed without any addressing overhead. While the DCT Tiler is a lossy mode, video quality remains high.

## 5. MULTISCALE DCT

The DCT Tiler experiment was our first experiment to exploit application-level semantics to utilize NDDI in a new, more sophisticated way. The next experiment aimed to leverage NDDI more fully to address the issue of rate control based on video content. The DCT Tiler has an adjustable quality factor, but our goal for the Multiscale DCT Tiler experiment was to try to exploit additional savings from more sophisticated configurations of NDDI.

The earlier scaled video player example in Figure 2 is a simple decoder that takes advantage of the Coefficient Plane mapping to scale a source video to 4x by mapping groups of four pixels from the display to a single pixel from the source. The same concept of scaling can be extended to the DCT Tiler such that pre-rendered basis functions are scaled to

cover larger than $8 \times 8$ blocks on the display. The obvious disadvantage here is that detail is lost when the source video is downscaled for transmission and then upscaled on the display. While this may be acceptable for areas of the frame without fine details, it is not a good scheme for those that do. Advantageously, NDDI affords a level of flexibility such that portions of the display can be rendered at a 1x scale while others are rendered at coarser scales.

NDDI does impose some design constraints that need to be considered. First, reconfiguring areas of the Coefficient Planes is not free. Therefore it is not advisable to do so often as it can impose a heavy penalty on a data transmission budget. To address this, we do not configure areas of the display at particular scales; instead we divide the Coefficient Planes among the different scales. This leads to the next constraint which is the limit of 64 Coefficient Planes. Losing even a few Coefficient Planes for a particular scale means that high frequency details cannot be rendered. In order to maximize quality as seen experimentally through PSNR, the 1x scale must retain a majority of the planes doling out only a few planes to the coarser scales.

Despite the constraints, the Multiscale DCT Tiler does show promise as a means to support rate control while maximizing quality. This is achieved primarily by updating large homogenous regions of the frame using the higher scales. Secondarily, the coarser scales serve as a prediction that can then be refined with the finer scales acting to address the residual differences. Figure 10 illustrates this using a 4x scale with only one Coefficient Plane. In such a configuration, the 1x scale only loses one plane and can therefore represent very fine details. However, the motivation of the Multiscale DCT Tiler is to experimentally find which configurations win out when the rate is reduced dramatically.

## 5.1 Multiscale Configuration

The Multiscale configuration is specified as a series of scales and number of planes dedicated to that scale. That number of planes is defined as the edge length in a square of basis functions. As an example, a 2 would indicate that 4 planes are reserved for that scale and that the DC coefficient and the AC coefficients at $(1,0)$, $(0,1)$, and $(1,1)$ are used. The choice of the shape is critical, because it allows the higher scales to create effective "checkerboard" outputs where each square of the rendered pattern provides a prediction for one or more finer blocks along perfect boundaries. When using a full $8 \times 8$ set of basis functions, it is likely that far fewer than 63 planes will be available. Therefore, the ordering remains zig-zag so that higher frequency AC coefficients are truncated first. The following illustrates the multiscale configuration used for the experiments.

- **1:8** - The only scale (1x) uses all 63 basis functions.
- **16:1, 1:8** - The first scale (16x) uses only the DC basis function. The remaining 62 are used for 1x.
- **2:2, 1:8** - The first scale (2x) uses a $2 \times 2$ square of basis functions. The remaining 59 are used for 1x.
- **4:3, 1:8** - The first scale (4x) uses 9 basis functions. The remaining 54 are used for 1x.
- **8:2, 4:2, 1:8** - The first two scales (8x, 4x) each use 4 basis functions. The remaining 55 are used for 1x.

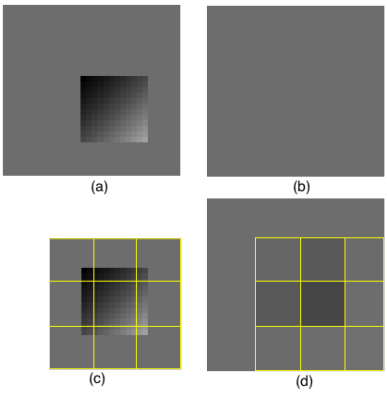


**Figure 10: Multiscale DCT example of 16 $8 \times 8$ blocks. (a) shows the image to be rendered consisting of mostly medium gray. If the first configured scale is 4x with only one plane, then (b) represents the contents of the display after that first layer is rendered. The next configured layer is 1x with the remaining 62 planes. After the first rendering, only 9 of the 16 blocks need to be updated. (c) shows the contents of those four blocks. Of the 9 blocks that are rendered at the 1x scale, 5 did not need to encode the DCT coefficient, because their DCT coefficients matched the coefficient from the 4x layer as seen in (d) which illustrates the DCT component from each of the 16 blocks at the 1x scale.**

## 5.2 Priority Updating Schemes

The Multiscale DCT experiments explore the different multiscale configurations in terms of performance when a transmission budget is enforced. The motivation for this change is that each additional set of planes dedicated to a new scale removes the highest frequency planes from the finest scale thus guaranteeing a loss in quality. Therefore, such a configuration must minimize the number of Scalers that are updated while sacrificing the minimal number of planes from the 1x scale. To minimize the Scalers sent over the wire, we employ four different priority schemes when choosing which Scalers to update.

When considering multiscale configurations and priority updating schemes, it is important to realize another constraint of NDDI. As seen with the DCT Tiler, it is possible to implement an NDDI command to update all of the Scalers for a "stack" of tiled regions of the Coefficient Plane, but it does incur a cost to address the stack and then to specify the size of the tiles and the stack size. Even going from a configuration of just 1x with a stack of one to three scales with three stacks incurs a penalty, but it is mitigated somewhat because the higher scales have fewer blocks. Our priority schemes seek to reduce the size of each stack either through leaving off Scalers for high frequency DCT coefficients from the "bottom" of the stack or the low frequency Scalers from the "top" of the stack as was seen with the example in Figure 10. We explored the following priority updating schemes each with a tunable variable *Delta* or *Plane Count* that is set for every frame in order to ensure that the budget is met.

**Trim with a Delta** The DCT Tiler trims the unchanged Scalers from the top and bottom of a stack of Scalers before sending them to the NDDI Display. When Trimming with

a Delta, then any Scaler where all three channels are within a Delta from the current Scaler is considered unchanged.

**Trim with a Plane Count** This form of trimming starts with the first unchanged Scaler from the top and trims off enough of the bottom Scalers to a stack height specified by the Plane Count.

**Snap to Zero with a Delta** In this scheme any Scaler with all three channels within a Delta from zero is set to zero. This is obviously effective for the high frequency AC coefficients at the bottom of the Scaler stack, but it is also beneficial for the DC and low frequency AC coefficients for the finer scales where coarser scales have produced a prediction that drives these coefficients near zero for the residual.

**Snap to Zero with a Plane Count** While still snapping to zero, this scheme starts with top-most non-zero scaler for each stack and zeroes out the bottom of the stack producing a stack height specified by the Plane Count.

## 5.3 Multiscale DCT Results

The multiscale configurations and priority schemes were evaluated using various ten-second clips with resolutions ranging from DVD to 4K. For each video source, we estimated the motion by calculating the optical flow using the Lucas-Kanade method with pyramids [12]. We then chose the ten second clips with the highest, lowest, and median flow. Each of the clips was tested with all twenty combinations of configuration and priority scheme using three budgets. The budgets were determined for each video clip using a calculation that used only 8, 4, and 2.5 bits per pixel for low, medium, and high compression. The calculations were adjusted to 4, 3, and 2 bits per pixel for 4K video to better suit the somewhat lower frequency data in the blocks.

Several of the clips exhibited a notable relationship between optical flow and the resulting PSNR. Figure 11 shows the median flow clip of a 720p movie. All twenty runs are shown using the medium (4 bpp) data budget. The flow for each frame is shown as an average that is normalized to $[0, 1000]$ and shown on a separate axis with a logarithmic scale. Nearly every run follows a similar trend; areas of high flow produce better PSNR. This is largely attributed to the motion blur associated with high flow sections. The blur effectively removes high frequency data from each block favoring a multiscale configuration that sacrifices several of the high frequency AC coefficients at the finest (1:8) scale. The low flow areas produced a much lower PSNR than we expected. This particular clip primarily consists of slow sweeping shots, and while the motion is not great this is enough to overcome the simple prediction residual model. Proper motion compensation would likely produce a superior result.

The clips with larger areas of static content performed much better. Figure 12 shows the median flow 4K clip. The top performing two priority schemes were chosen and all three compression settings (4 bpp, 3 bpp, and 2 bpp) were used for each priority scheme. This particular clip is a movie trailer that contains many rapid scene changes with scenes ranging from dialog to high action. Additionally the scene transitions are marked with large black screens with little text. Some of the previous trend is seen with high flow sections yielding higher PSNR, but there was lessened reduction in PSNR for the low flow sections because the flow did not uniformly affect the entire scene. The clip starts with a black screen displaying an actor's name in the center. It then fades to a head shot punctuated later by a fade through

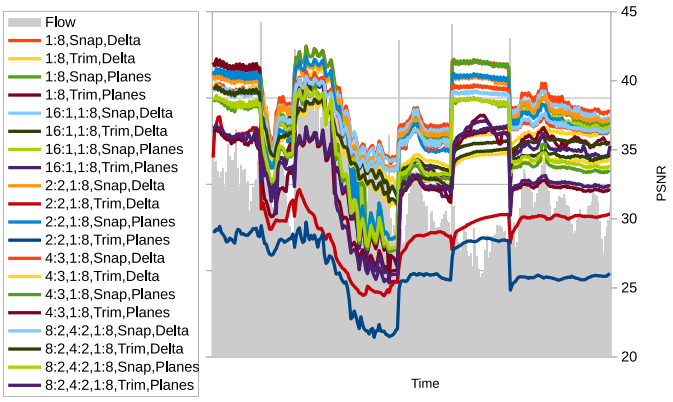


**Figure 11: PSNR of all configuration and priority schemes contrasted to optical flow using the medium budget constraint on the Limitless 720p median flow clip.**

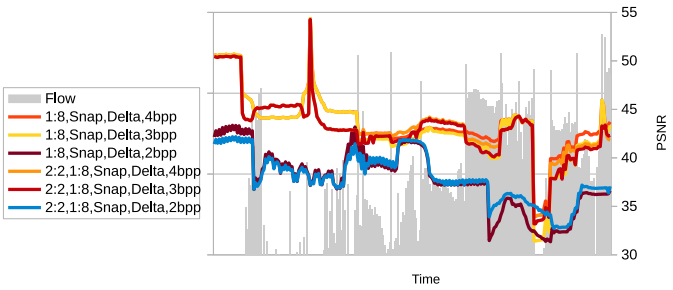


**Figure 12: PSNR of best configuration and priority schemes contrasted to optical flow using all three budge constraints on the Elysium 4K median flow clip.**

black that produces the very high peak with a PSNR of nearly 64 for the low and medium compression.

The average PSNR for each clip is shown in Tables 2 and 3 alongside the best configuration. For the high and medium data budgets a basic 1:8 scale configuration performed best. For the high data budget the priority schemes performed very similarly, whereas Snap Delta performed best for medium data budgets. Perhaps most interesting, the 2:2 1:8 scale configuration and Snap Delta performed the best for all of the clips when using the low data budget.

The DCT Tiler uses the data budgets as upper bounds when determining either the plane count or delta for each scale. Therefore the data actually transmitted with an 8 bpp budget may not be exactly double that of a 4 bpp budget. Figures 13 and 14 show the total number of bytes sent over the NDDI Link as a ratio to a 60 Hz signal. The larger formats clips generally perform better. The trailer clips (captain and elysium) perform exceptionally well in areas of low flow due to large static frames containing informational text for the audience. In fact, for the 4K elysium clip a majority of the frames were rendered under the tightest budget while still updating all changed Scalers.

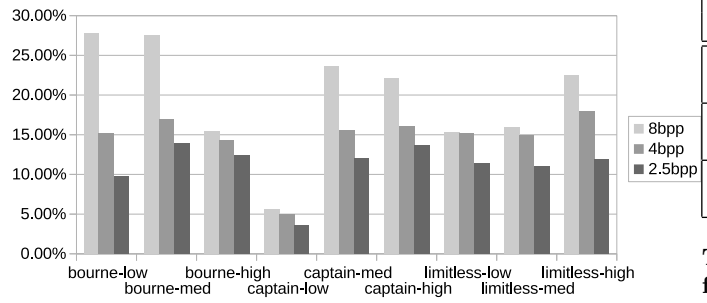


**Figure 13: Video results for the DVD, 720p, and 1080p 10s clips using the medium data budget and the configuration and priority scheme that performed best. As with before, the results are a ratio of the byte transmitted over the wire compared to a 60Hz.**

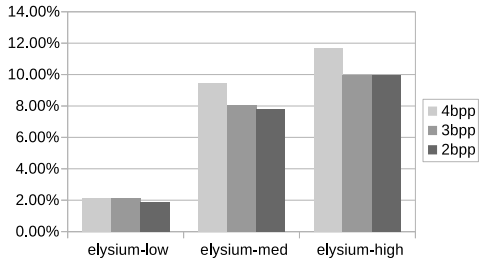


**Figure 14: Video results for 4K 10s clips using the medium data budget and the configuration and priority scheme that performed best. As with before, the results are a ratio of the byte transmitted over the wire compared to a 60Hz.**

| Video / Flow | Budget | | |
|---|---|---|---|
| | 8 bpp | 4 bpp | 2.5 bpp |
| bourne DVD low | 35.75 - 1:8 trim delta | 33.79 - 1:8 trim delta | 31.99 - 2:2 1:8 snap delta |
| bourne DVD median | 38.64 - 1:8 snap planes | 36.57 - 1:8 snap delta | 34.91 - 2:2 1:8 snap delta |
| bourne DVD high | 44.01 - 1:8 trim planes | 43.69 - 1:8 snap delta | 41.11 - 2:2 1:8 snap delta |
| captain 720 low | 41.14 - 1:8 trim delta | 40.44 - 2:2 1:8 snap delta | 38.98 - 2:2 1:8 snap delta |
| captain 720 median | 39.94 - 1:8 snap planes | 38.67 - 1:8 snap delta | 37.43 - 2:2 1:8 snap delta |
| captain 720 high | 40.52 - 1:8 snap planes | 39.79 - 1:8 snap delta | 38.72 - 2:2 1:8 snap delta |
| limitless 1080 low | 41.62 - 1:8 snap delta | 41.53 - 1:8 snap planes | 39.72 - 1:8 snap delta |
| limitless 1080 median | 41.33 - 1:8 snap delta | 41.06 - 1:8 snap delta | 39.28 - 1:8 snap delta |
| limitless 1080 high | 39.52 - 1:8 trim delta | 38.79 - 1:8 snap delta | 37.51 - 16:1 1:8 snap delta |

**Table 2: PSNR and Optimal configuration / scheme for each of the low, medium, and high flow clips. The data budgets are 8, 4, and 2.5 bits per pixel.**

| Video / Flow | Budget | | |
|---|---|---|---|
| | 4 bpp | 3 bpp | 2 bpp |
| elysium 4K low | 48.78 - 1:8 snap delta | 48.79 - 1:8 trim planes | 48.32 - 2:2 1:8 snap delta |
| elysium 4K median | 43.56 - 1:8 snap delta | 43.03 - 1:8 snap delta | 38.06 - 2:2 1:8 snap delta |
| elysium 4K high | 43.14 - 1:8 snap delta | 42.47 - 1:8 snap delta | 41.35 - 2:2 1:8 snap delta |

**Table 3: PSNR and Optimal configuration / scheme for each of the low, medium, and high flow 4K clips. The data budgets are 4, 3, and 2 bits per pixel.**

# 6. RELATED WORK

In defining the n-Dimensional Display Interface, we sought to redefine a new narrow waist while avoiding the "Wheel of Reincarnation" [14] by just raising the abstraction. Some of the earliest graphics interfaces have tackled the rendering pipeline with highly parallel logic. The framebuffer has survived since the advent of high-density integrated memory in the 1970's with the SuperPaint [18] system from Xerox Parc. As one of the earliest 3D systems, Pixel Planes [8] used programmable logic to realize equations used to render polygons and spheres as well as compute advanced graphics techniques such as shadows, transparency, and anti-aliasing. As 3D rendering became more prevalent, the parallelism became more evident [2]. Slowly the computation moved further from the display. Today GPUs embody much of this parallelism into dedicated hardware distinct from the display. They perform their operations and reduce the result to a single framebuffer that is serialized and sent over the wire.

Today, growing display resolutions and transmission media have led to numerous new standards and technologies to better cope. Digital standards like DVI [20] [4], HDMI [6] [10], and DisplayPort [22] have coped well enough, but they have not used parallelism to address the problem of pixel transmission instead relying on digital compression. Research into large-scale displays and display walls is tackling the problem through more explicit parallelism. IBM's Scaleable Graphics Engine (SGE) [17] introduced a hardware framebuffer allowing separate nodes to render their content. With such high resolutions, collaboratively rendering display content proved promising in systems such as Lightning-2 [19], Chromium [11], SAGE [3], Garuda [16], and Equalizer [5]. These systems each seek to composite the video output from several sources and render it to a single large display or multiple tiled displays. Despite their efficiency, each system still talks to the attached displays as a framebuffer forcing the composited video to be then split amongst displays. A single large NDDI Display can easily be leveraged allowing the sources to write to their own regions of the Frame Volume and Coefficient Planes.

Research into acceleration of large format video on displays has focused primarily on embedding codecs into displays in support of signaling standards such as DPVL [21]. However, certain aspects of video decoding are "embarrassingly parallel" and so GPUs have been leveraged to aid in video decoding [9]. While embedding a GPU on a display can be promising, it raises the abstraction significantly compared to NDDI, which provides basic DCT support at a much lower abstraction.

Perhaps the most similar research in the area is that of the embedded display processor described in the Embedded Function Composition system [23]. This research addresses many of the same challenges as NDDI with a much more sophisticated approach resembling older systems such as Pixel Planes. While is may prove to be more flexible than NDDI, the abstraction is again quite a bit higher and is therefore less suitable as a "narrow waist".

# 7. CONCLUSIONS AND FUTURE WORK

In prior experiments, NDDI provided little benefit for full screen video when the NDDI Display was configured without regard to application-level framing. Our extension to the NDDI architecture for blending was highly effective for enabling considerable data transmission reduction for full screen video by blending pre-rendered basis functions for each of the $8 \times 8$ blocks. With these new extensions, NDDI proved versatile for implementing rate control while maximizing quality, especially for very high resolution 4K video. Despite only using intra-frame data rate control, we were able to maintain strong PSNR under much smaller budgets than any of the earlier experiments. In doing so, we discovered trends in the Multiscale DCT Tiler performance given particular video frame content and types of motion. Future work will employ more realistic inter-frame rate control while supporting mixed media display content as we move into even more specialized uses cases with greater application-level framing.

# 8. REFERENCES


[1] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. *ACM SIGCOMM Computer Communication Review*, 20(4):200–208, 1990.

[2] T. W. Crockett. An introduction to parallel rendering. *Parallel Computing*, 23(7):819–843, July 1997.

[3] M. Deering and D. Naegle. The SAGE Graphics Architecture. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 683–692, New York, NY, USA, 2002. ACM.

[4] Digital visual interface dvi revision 1.0. Digital Display Working Group, April 1999.

[5] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: a scalable parallel rendering framework. In *ACM SIGGRAPH ASIA 2008 courses on - SIGGRAPH Asia '08*, SIGGRAPH Asia '08, pages 1–14, New York, New York, USA, 2008. ACM Press.

[6] P. C. Electronics, B. V. International, S. Image, S. Corporation, and T. Corporation. High-Definition Multimedia Interface, 2006.

[7] C. D. Estes and K. Mayer-Patel. The n-Dimensional Display Interface A More Elastic Narrow Waist for the Display Pipeline. In *MMSYS*, 2012.

[8] H. Fuchs, J. Goldfeather, J. P. Hultquist, S. Spach, J. D. Austin, F. P. Brooks Jr., J. G. Eyles, and J. Poulton. Fast Spheres, Shadows, Textures, Transparencies, and Imgage Enhancements in Pixel-planes. *SIGGRAPH Comput. Graph.*, 19(3):111–120, July 1985.

[9] B. Han and B. Zhou. Efficient video decoding on GPUs by point based rendering. *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware - GH '06*, page 79, 2006.

[10] High-definition multimedia interface specification version 1.3a. HDMI Founders, November 2006.

[11] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*, SIGGRAPH '02, page 693, New York, New York, USA, 2002. ACM Press.

[12] B. D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. *Imaging*, 130(x):674–679, 1981.

[13] U. McMillan, L. (Sun Microsystems Inc., Research Triangle Park, NC and L. Westover. A forward-mapping realization of the inverse discrete cosine transform. In *Data Compression Conference*, pages 219 – 228, Snowbird, UT, USA, 1992.

[14] T. H. Myer and I. E. Sutherland. On the design of display processors, 1968.

[15] M. Nelson. *The Data Compression Book*. M&T Books, San Mateo, CA, 1992.

[16] H. Nirnimesh (Institute of Information Technology, P. Harish, and P. Narayanan. Garuda: A Scalable Tiled Display Wall Using Commodity PCs. *Visualization and Computer Graphics, IEEE Transactions on*, 13(5):864 – 877, 2007.

[17] K. A. Perrine and D. R. Jones. Parallel graphics and interactivity with the scaleable graphics engine. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM) - Supercomputing '01*, Supercomputing '01, pages 5–5, New York, New York, USA, 2001. ACM Press.

[18] R. Shoup. SuperPaint: an early frame buffer graphics system. *IEEE Annals of the History of Computing*, 23(2), 2001.

[19] G. Stoll, M. Eldridge, D. Patterson, A. Webb, S. Berman, R. Levy, C. Caywood, M. Taveira, S. Hunt, and P. Hanrahan. Lightning-2: A High-performance Display Subsystem for PC Clusters. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, number August in SIGGRAPH '01, pages 141–148, New York, NY, USA, 2001. ACM.

[20] B. Sung, S. H. Hwang, and V. Da Costa. DVI: A Standard for the Digital Monitor Interface, 1999.

[21] Vesa dpvl software interface standard: Version 1. Video Electronics Standards Association, February 2006.

[22] Vesa displayport: Version 1.1. Video Electronics Standards Association, April 2007.

[23] T. Whitted, J. Kajiya, E. Ruf, and R. Bittner. Embedded Function Composition. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 47–50, New York, NY, USA, 2009. ACM.