Exploiting Human Color Discrimination for Memory- and Energy-Efficient Image Encoding in Virtual Reality

Nisarg Ujjainkar
nujjaink@ur.rochester.edu
University of Rochester
Rochester, NY, USA

Ethan Shahan
eshahan@u.rochester.edu
University of Rochester
Rochester, NY, USA

Kenneth Chen
kennychen@nyu.edu
New York University
New York, NY, USA

Budmonde Duinkharjav
budmonde@nyu.edu
New York University
New York, NY, USA

Qi Sun
qisun@nyu.edu
New York University
New York, NY, USA

Yuhao Zhu
yzhu@rochester.edu
University of Rochester
Rochester, NY, USA

Abstract

Virtual Reality (VR) has the potential of becoming the next ubiquitous computing platform. Continued progress in the burgeoning field of VR depends critically on an efficient computing substrate. In particular, DRAM access energy is known to contribute to a significant portion of system energy. Today's framebuffer compression system alleviates the DRAM traffic by using a numerically lossless compression algorithm. Being numerically lossless, however, is unnecessary to preserve perceptual quality for humans. This paper proposes a perceptually lossless, but numerically lossy, system to compress DRAM traffic. Our idea builds on top of long-established psychophysical studies that show that humans cannot discriminate colors that are close to each other. The discrimination ability becomes even weaker (i.e., more colors are perceptually indistinguishable) in our peripheral vision. Leveraging the color discrimination (in)ability, we propose an algorithm that adjusts pixel colors to minimize the bit encoding cost without introducing visible artifacts. The algorithm is coupled with lightweight architectural support that, in real-time, reduces the DRAM traffic by 66.9% and outperforms existing framebuffer compression mechanisms by up to 20.4%. Psychophysical studies on human participants show that our system introduce little to no perceptual fidelity degradation.

ACM Reference Format:
Nisarg Ujjainkar, Ethan Shahan, Kenneth Chen, Budmonde Duinkharjav, Qi Sun, and Yuhao Zhu. 2024. Exploiting Human Color Discrimination for Memory- and Energy-Efficient Image Encoding in Virtual Reality. In *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3617232.3624860

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0372-0/24/04.
https://doi.org/10.1145/3617232.3624860

1 Introduction

Virtual Reality (VR) has the potential of becoming the next ubiquitous computing platform, after PCs and smartphones, revolutionizing a wide variety of domains such as healthcare [7], education [9], remote communication [44, 49], professional training [28], and industrial design [23].

Continued progress in the burgeoning field of VR depends critically on an efficient computing substrate, driven by the ever-growing requirement of immersive user experience and the miniaturization of device form factors. DRAM communication energy is known to contribute significantly to the system energy consumption. Recent studies show that DRAM energy alone can consume upward of 30% of the total system energy consumption during VR video rendering [27, 77]. The DRAM bottleneck will only become worse in the future with users questing for higher resolution and frame rate.

An effective approach to reduce DRAM traffic is framebuffer compression, which is universally implemented in modern mobile SoCs for compressing any traffic in and out of the DRAM. A classic example is the Arm Frame Buffer Compressions (AFBC) technology, which is now in almost all of Arm's GPU, Video Codec, and Display Controller IPs [3].

Idea. Today's framebuffer compression algorithm is numerically lossless. Being numerically lossless is, however, unnecessary to preserve perceptual fidelity: more compression opportunities arise when we turn our attention to *perceptual lossless*. Long-established psychophysical studies show that humans cannot discriminate colors that are close to each other [36, 68]. Informally, this means that many colors, while differing in RGB values, are perceptually indistinguishable and thus can be encoded together — a previously under-exploited opportunity for real-time image encoding.

Critically, the discrimination ability becomes even weaker (i.e., more colors are indistinguishable) in our *peripheral* vision as objects move away from fixation [14, 22, 29]. The eccentricity-dependent weakening of color discrimination provides further opportunities for DRAM traffic compression: VR displays, to provide immersive experiences, have a wide Field-of-View (FoV) of about 100°; above 90% of a frame's pixels are in the peripheral vision (outside 20 °) [10, 45].

Design. Leveraging the unique color discrimination (in)ability of human visual system, we propose a new image compression algorithm for immersive VR systems. We precisely formulate the color perception-aware encoding as a constraint optimization problem. The formulation is non-convex and requires iterative solvers that are not amenable to real-time execution. Leveraging empirical observations of human color discrimination abilities, we introduce a set of principled relaxations, which transform the compression problem into a convex optimization with an analytical solution.

The analytical solution, while avoiding iterative solvers, is still compute intensive and slow to execute in real-time. Implemented as a GPU shader executing on the Adreno 650 GPU in Oculus Quest 2, a widely used mobile VR headset, the compression algorithm runs in a mere 2 FPS. We propose lightweight hardware extensions for our encoding and decoding algorithms. The new hardware exploits the inherent task-level and pipeline-level parallelisms in the algorithms and can be readily combined with existing Base-Delta (BD) encoding without changing the decoding hardware at all.

Results. We implement our architectural extensions in RTL and synthesize the design using a TSMC 7 nm process node. The compression algorithm reduces the memory traffic by 66.9% compared to uncompressed images and by up to 20.4% compared to the state-of-the-art real-time framebuffer compression [76]. We conduct IRB approved human subject study on 11 participants. Results suggest that our compression algorithm brings little visible artifacts to users. In summary, this paper makes the following contributions:

- We propose an image encoding scheme to reduce DRAM traffic in mobile VR systems. The scheme leverages the eccentricity-dependent color discrimination (in)ability of human visual systems.
- We show that the new encoding scheme can be formulated as a convex optimization problem with an analytical solution.
- We propose lightweight and modular hardware support to enable real-time encoding.
- ASIC synthesis and human subject studies show that the new encoding scheme reduces the DRAM traffic by 66.9% with little to no subjective perceptual quality degradation.

The rest of the paper is organized as follows. Sec. 2 introduces the background. Sec. 3 describes our key compression algorithm. Sec. 4 introduces the co-designed hardware architecture. Sec. 5 discusses the experimental methodology, followed by the evaluation results in Sec. 6. We relate our work to prior art in Sec. 7 and conclude in Sec. 8.

2 Background and Motivation

We first introduce the background of human color perception and its eccentricity dependence, which form the psychophysical basis for our compress algorithm (Sec. 2.1). We then describe today's real-time frame compression algorithm, which forms an important baseline for our algorithm (Sec. 2.2).

2.1 Eccentricity-Dependent Color Perception

Colors and Color Spaces. In a typical rendering pipeline, a color is usually described in the linear RGB space with three channels; each channel is a floating point number between 0 and 1. For output encoding, each channel in the linear RGB color space is transformed to the common sRGB color space, where each channel is an 8-bit integer between 0 and 255. This transformation is non-linear, called gamma encoding, and is described by the following function f_{s2r}, where $x \in [0, 1]$ represents a linear RGB channel value [24, 62]:

$$f_{s2r}(x) := \begin{cases} \lfloor 12.92x \rfloor & x \le 0.0031308 \\ \lfloor 1.055x^{1/2.4} - 0.055 \rfloor & x > 0.0031308 \end{cases} \tag{1}$$

Psychophysical studies on color discrimination commonly operate in the DKL color space [19, 29], mainly because the DKL space models the opponent process in the human visual system. The DKL space is a linear transformation away from the linear RGB color space:

$$[R, G, B]^T = \mathrm{M}_{\mathrm{RGB2DKL}} [K_1, K_2, K_3]^T \tag{2}$$

where $[R, G, B]$ is the color in the linear RGB space, $[K_1, K_2, K_3]$ is the color in the DKL space, and $\mathrm{M}_{\mathrm{RGB2DKL}}$ is a 3×3 constant matrix (with the same coefficients, $[[0.14, 0.17, 0.00], [-0.21, -0.71, -0.07], [0.21, 0.72, 0.07]]$, as in Duinkharjav et al. [22]).

Color Discrimination. It is well-established that humans can not discriminate between colors that are close to each other [36, 68]. For instance, Fig. 1 shows four colors that have different sRGB values but appear to be the same.

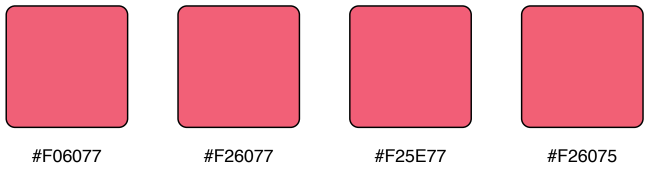


Fig. 1. Human visual system can not discriminate colors that close to each other. These four colors differ in tristimulus values, but appear to be the same color.

More formally, given a reference color κ, there exists a *set* of colors $\mathcal{E}_\kappa$, in which all the colors are perceptually indistinguishable from κ. In a linear color space such as DKL and RGB, the set of equi-appearance colors in $\mathcal{E}_\kappa$ form an *ellipsoid*, whose center is κ [41]. In the literature, such an ellipsoid is called a *discrimination ellipsoid* [69].

Eccentricity Dependence. Critically, human color discrimination ability is weaker in the peripheral vision [14, 22].

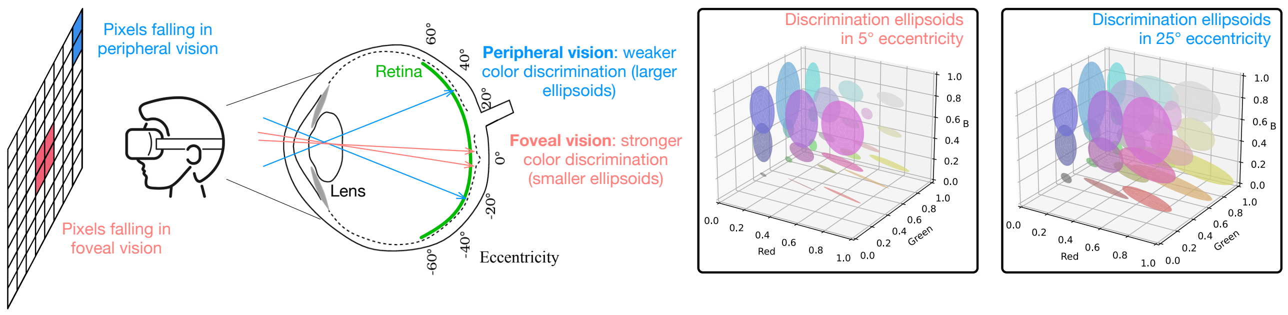

Pixels falling in peripheral vision
Pixels falling in foveal vision
Retina
Lens
Eccentricity
60°
40°
20°
0°
-20°
-40°
-60°
Peripheral vision: weaker color discrimination (larger ellipsoids)
Foveal vision: stronger color discrimination (smaller ellipsoids)
Discrimination ellipsoids in 5° eccentricity
Discrimination ellipsoids in 25° eccentricity
Red
Green
B

Fig. 2. Color discrimination is eccentricity dependent. The discriminative ability is weaker as the eccentricity increases. As a result, the sizes of the discrimination ellipsoids increase with the eccentricity. The two plots on the right show the discrimination ellipsoids under a 5° and a 25° eccentricity, respectively, in the linear RGB color space (i.e., sRGB normalized to [0, 1] without gamma [24, 62]). The discrimination ellipsoids in each plot are shown for 27 colors uniformly sampled in the linear RGB color space between [0.2, 0.2, 0.2] and [0.8, 0.8, 0.8].

That is, for a color κ, its discrimination ellipsoid $\mathcal{E}_\kappa$ is larger, i.e., includes more indistinguishable colors, as κ moves away from one's fixation. Fig. 2 shows two figures that plot the discrimination ellipsoids under a 5° and a 25° eccentricity, respectively, in the linear RGB color space. Eccentricity is the angle from the center of the retina, a.k.a., current fixation or "fovea". The ellipsoids in the 25° plot are larger than those in the 5° plot, suggesting that the color discrimination ability is weaker in peripheral vision.

Color discrimination becomes weaker in the visual periphery for three reasons. First, the receptive field (RF) sizes of Retinal Ganglion Cells (RGCs) increase with eccentricity, a result of larger dendritic fields [17, 52] and sparser RGC density in periphery [15]. A large RF means that a RGC integrates signals from a larger spatial area, leading to more blurring in the (spatial) frequency domain. Second, cone cells (which are photoreceptors responsible for vision under normal daylight) become larger in size as eccentricity increases [16], also contributing blurring in spatial frequency. Finally, the distribution of cone cells on our retina is extremely non-uniform: over 95% of the cone cells are located in the central region of the retina (i.e., fovea) with an eccentricity of below 5 ° [16, 56]. The density of the cone cells decreases drastically in the visual periphery, which is, thus, significantly under-sampled spatially.

The full color discrimination function Φ, expressed below, is thus parameterized by both the reference color κ and the eccentricity $\mathbf{e}$:

$$\Phi : (\kappa, \mathbf{e}) \mapsto (a, b, c) \tag{3}$$

where (a, b, c) represents the semi-axes lengths of the discrimination ellipsoid belonging to color κ at an eccentricity $\mathbf{e}$ in the DKL color space [19], a common color space for color perception experiments. Given (a, b, c), $\mathcal{E}_\kappa$, the discrimination ellipsoid of color κ in the DKL space, is given by:

$$\frac{(x-\kappa_1)^2}{a^2} + \frac{(y-\kappa_2)^2}{b^2} + \frac{(z-\kappa_3)^2}{c^2} = 1 \tag{4}$$

where $(\kappa_1, \kappa_2, \kappa_3)$ represent the three channels of the color κ.

The function Φ can be implemented using a Radial Basis Function (RBF) network [22], which is extremely efficient to implement on GPUs in real time. In our measurement on Oculus Quest 2 VR headset using Oculus' OVR Metrics Tool [5], evaluating RBF network runs in 72 FPS, matching the display refresh rate while consuming sub 1 mW power.

AR and VR headsets, in providing an immersive experience, usually have a wide FoV that is above 100°. Therefore, the vast majority of the pixel colors will fall in the peripheral vision. The eccentricity-dependent color discrimination (in)abilities of human visual system gives opportunities to better image compression that this paper exploits.

2.2 Real-Time Frame Compression

DRAM Traffic. A variety of data communication traffics occur on a VR system, as illustrated in Fig. 3, such as the traffic through DRAM, the display interface, and the wireless communications with a remote rendering server. This paper focuses on reducing the DRAM traffic, which occurs when the different Intellectual Property (IP) blocks in the SoC communicate with each other during rendering.

Each frame, the GPU writes the frame data to the frame buffer in the GPU, which are then read by the display controller. It is these DRAM traffics (i.e., GPU $\leftrightarrow$ frame buffer $\leftrightarrow$ DRAM controller) that this paper focuses on reducing. When rendering a VR (360°) video, additional DRAM traffics occur between the network interface controller, the video codec, and the GPU [38]. While not explicitly targeted in this paper, these traffics can also potentially be reduced by our compression algorithm, especially in scenarios where

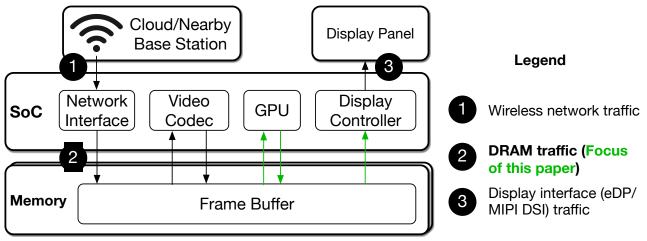

Cloud/Nearby Base Station
Display Panel
Legend
SoC
Network Interface
Video Codec
GPU
Display Controller
Memory
Frame Buffer
1 Wireless network traffic
2 DRAM traffic (Focus of this paper)
3 Display interface (eDP/ MIPI DSI) traffic

Fig. 3. Different types of data communication traffic in a VR system. This paper focuses on reducing DRAM traffic.

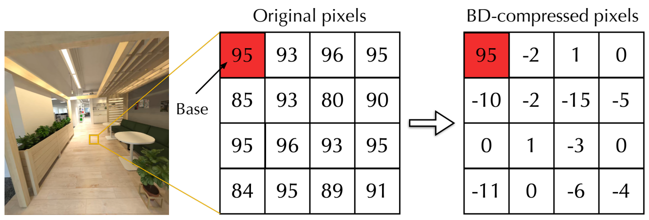


Fig. 4. Base + Delta (BD) compression, which works in the sRGB color space. For each pixel tile (4×4 here), we find a base pixel (95 here), and calculate the Δ of all other pixels from the base pixel. The Δ are smaller in magnitude and thus require fewer bits to encode. The same compression strategy is applied to all three color channels.

remotely rendered frames are transmitted one by one (rather than as a video) [31, 35].

Reducing DRAM traffic is critical. It is well-established that data transfer and memory access energy is known to far out-weigh the energy consumption of computation. For instance, compared to a Multiple-Accumulate (MAC) operation on 1-Byte fixed-point data, transferring one Byte of information through DRAM consumes 800 × [39, 40] higher energy. Reducing DRAM traffic in a visual computing system has been a main research focus in recent years [27, 34, 76].

Framebuffer Compression Algorithms. An effective and commonly used approach to reduce DRAM traffic in a rendering system is framebuffer compression, which compresses and uncompresses every frame in and out of the DRAM. To ensure a low *per-frame* latency, compression in VR must be done on a per-frame basis, precluding video compression methods such as H.265/VP5, which necessarily require buffering a sequence of frames before compression [50, 51]. Offline image compression methods such as JPEG and PNG are rarely used in framebuffer compression as they are too compute-intensive. For instance, JPEG requires chroma subsampling, transforming images to a frequency space followed by quantization and Huffman encoding [63].

Today's framebuffer compression methods universally use a much faster base+delta (BD) strategy. Fig. 4 uses a simple example to illustrate the basic idea behind BD, which compresses each color channel and each pixel tile individually. The tile size in Fig. 4 is 4×4. In each tile, BD chooses a base pixel and then calculates the Δs/offsets between all other pixels and the base pixel. In the example of Fig. 4, the base pixel is the first pixel. The Δs will necessarily have smaller magnitudes compared to the original pixel values and, thus, require fewer bits to encode.

The BD compression algorithm is lightweight: it works completely in the image space, as opposed to the frequency domain which requires an additional, compute-intensive transformation (e.g. Fast Fourier Transform or Discrete Cosine Transformation); it requires only fixed-point addition arithmetics; it is also embarrassingly parallel. Therefore, the basic BD strategy is universally implemented in today's mobile SoCs for compressing any traffic in and out of the DRAM. A classic example is the Arm Frame Buffer Compressions (AFBC) technology, which is now in almost all of Arm's GPU, Video Codec, and Display Controller IPs [3].

3 Color Perception-Aware Compression

This section introduces a color perception-aware image encoding and decoding algorithm. We start by describing the high-level ideas (Sec. 3.1), followed by a precise problem formulation in the form of constraint optimization (Sec. 3.2). We then show how this optimization problem has an analytical solution when relaxed to a convex problem (Sec. 3.3). We then describe the full compression algorithm (Sec. 3.4).

3.1 Key Ideas

The basic BD algorithm is numerically lossless. Our observation is that numerically lossless compression is unnecessary to preserve perceptual equivalence — because of the inherent the color discrimination (in)ability of human visual system.

Intuition. The basic BD algorithm encodes all the Δs in a tile (off of a base pixel) rather than the original pixel values. Thus, to improve the compression ratio over BD we must reduce the magnitude of the Δs, which, intuitively, requires bringing pixels *more similar* to each other.

Under a numerically lossless constraint, however, the Δs between pixels are fixed. Our idea is to relax the constraint from numerical lossless to *perceptually lossless*. In this way, we could adjust pixel color values, as long as each pixel color does not go beyond its discrimination ellipsoid, to minimize the total number of bits required to encode the Δs. This encoding is numerically lossy as we intentionally change the color values, but will preserve the perceptual quality.

An Example. More concretely, consider the example in Fig. 5, which shows 16 pixels in a tile on an axis. The number of bits required to encode the entire tile is (ignoring any metadata for now):

$$B = B_0 + N \times B_D \tag{5}$$

$$B_0 = 8, N = 15, B_D = \lfloor log_2(Max - Min + 1) \rfloor \tag{6}$$

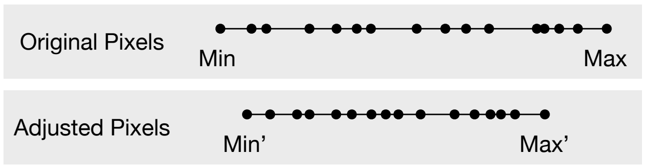


Fig. 5. An intuition illustration of our perceptual-aware compression, where pixel values are adjusted to be more similar to each other by leveraging the inherent human color discrimination thresholds.

where B_0 being 8 denotes that we need 8 bits to encode a base pixel (assuming the common 8-bit per-channel encoding), and N being 15 denotes that there are 15 other pixels. B_D denotes the number of bits required to encode the Δ of each of the 15 non-base pixels.

The minimum value of B_D occurs when the base pixel is chosen to be within $[Min, Max]$, in which case $B_D = \lfloor log_2(Max - Min + 1) \rfloor$. This is because the number of bits to encode each Δ must be the same[1], so we must accommodate the *largest possible* Δ, which is the difference between the maximum and minimum pixels in the tile. Therefore, to improve compression ratio we must reduce $(Max - Min)$.

The bottom example in Fig. 5 illustrates what would happen when we relax the compression constraint to be perceptually lossless. The adjusted pixel values deviate from the original values, but as long as they still within the respective ellipsoids, $(Max - Min)$ is reduced without affecting perceptual quality.

It is worth noting that to obtain the highest compression rate it is necessary to adjust interior pixels, as is the case in this example. The central challenge we address in this paper is how to design a principled algorithm that maximizes the bit reduction while being lightweight to execute in real time.

3.2 Problem Formulation

Our compression algorithm works on top of the baseline BD algorithm. Our goal is to adjust pixel colors to minimize the bit-length required to encode the Δs in a tile. The adjusted pixel tile then goes through any existing BD compression method. Critically, color adjustment must not violate the perceptual constraints. Therefore, we formulate our compression as a constraint optimization problem:

$$\underset{\mathbf{p}}{\text{argmin}} \sum_{C \in \{R,G,B\}} log_2 \lfloor max\{f_{s2r}(\mathbf{p}^C)\} - min\{f_{s2r}(\mathbf{p}^C)\} + 1 \rfloor, \tag{7a}$$

$$\text{where } \mathbf{p} := [p_0,\ p_1,\ \cdots,\ p_{N-1}], \tag{7b}$$

$$\mathbf{p}^C := [p_0^C,\ p_1^C,\ \cdots,\ p_{N-1}^C],\ C \in \{R, G, B\} \tag{7c}$$

$$s.t.\ \forall p_i \in \mathbf{p}\ p_i \in \mathcal{E}_{p_i} \tag{7d}$$

where $\mathbf{p}$ is the optimization variable, which is the collection of N pixels in a tile (Equ. 7b); p_i^C denotes channel C (R, G, or B) of i-th pixel in the linear RGB space.

The constraints (Equ. 7d) provide the (convex) ellipsoid boundary for each pixel to move while maintaining perception quality. $f_{s2r}(\cdot)$ is the non-linear transformation from RGB to sRGB space (Sec. 2.1), which is ultimately where bit encoding takes place. The objective function (Equ. 7a) minimizes the bit cost for encoding the Δs across all channels (it is a constant cost to encode the base pixel, e.g., 8 in the common sRGB encoding). This optimization formulation is applied to each pixel tile independently.

Unfortunately, this optimization problem is impractical to be solved in real-time, because the objective function is non-convex due to the non-linearity of min, max, floor, and $f_{s2r}(\cdot)$. Empirically, we also find that the popular solvers in Matlab spend hours while still being stuck in local optima.

Relaxation. We introduce two relaxations that turn the problem into a convex optimization. Critically, while general convex optimization requires iterative solvers (e.g., gradient descent or Newton's method [11]), our relaxed problem is one such that it has an analytical solution. The relaxations keep the same constraints as before (Equ. 7d) and, thus, still enforce the perceptual quality.

The first relaxation is based on the empirical observation that most discrimination ellipsoids are elongated along the either the Red or the Blue axis. See the discrimination ellipsoids in Fig. 2 for an illustration. This makes sense as human visual perception is most sensitive to green lights [54, 69] and, thus, has the least "wiggle room" along the Green axis.

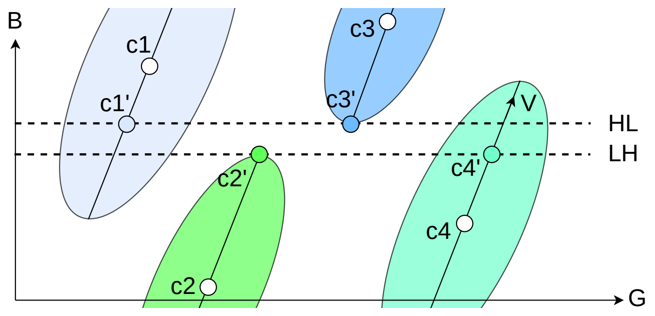
Our idea thus is to, instead of minimizing the bit costs across *all* three axes, minimize along *only* the Red or the Blue axis (while still having the flexibility of adjusting all the channels of all the pixels in a tile). Using the Blue axis an example, this relaxation yields following new objective function in Equ. 8a:

$$\underset{\mathbf{p}}{\text{argmin}}\ log_2 \lfloor max\{f_{s2r}(\mathbf{p}^B)\} - min\{f_{s2r}(\mathbf{p}^B)\} + 1 \rfloor, \tag{8a}$$

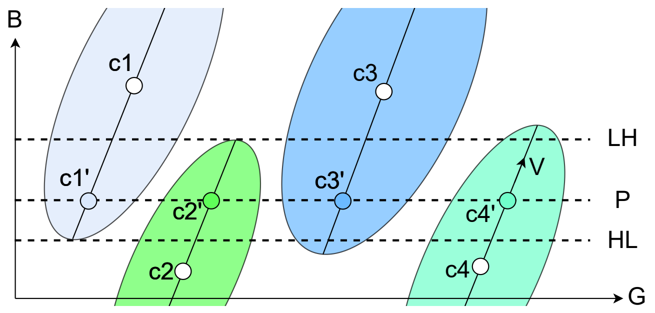
$$\Rightarrow \underset{\mathbf{p}}{\text{argmin}}\ max\{f_{s2r}(\mathbf{p}^B)\} - min\{f_{s2r}(\mathbf{p}^B)\}, \tag{8b}$$

$$\overset{\sim}{\Rightarrow} \underset{\mathbf{p}}{\text{argmin}}\ max\{\mathbf{p}^B\} - min\{\mathbf{p}^B\}. \tag{8c}$$

[1]It is possible, but uncommon, to vary the number of bits to encode the Δs in a tile with more hardware overhead. Following prior work [76], this paper assumes that one single bit-length is used to encode all Δs in a tile. We consider variable bit-length an orthogonal idea to this paper.



(a) Case 1: $HL > LH$.



(b) Case 2: $HL < LH$.

Fig. 6. The two cases in adjusting color values to minimize the Δ along the Blue axis. For simplicity, we draw the ellipsoids in the B-G plane. The empty markers (C_0, C_1, C_2, C_3) denote the original colors and the solid markers (C'_0, C'_1, C'_2, C'_3) denote the adjusted colors. In both cases, colors are adjusted along the extrema vector **V**.

Second, the objective function in Equ. 8a can be transformed to Equ. 8b without sacrificing solution optimality, because $log_2\lfloor\cdot\rfloor$ is monotonically non-decreasing. We then remove the non-linear RGB to sRGB transformation function $f_{s2r}(\cdot)$. This removal does not preserve the solution optimality, but gives us a convex objective function in Equ. 8c.

Proof of Convexity. Let the objective function $max\{\mathbf{x}\} - min\{\mathbf{x}\}$ be $g(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$. To prove $g(\mathbf{x})$ is convex, we must show: $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^N$ and $t \in [0,1]$, $g(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq tg(\mathbf{x}_1) + (1-t)g(\mathbf{x}_2)$.

Proof. Observe that: $g(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) := max(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) - min(t\mathbf{x}_1 + (1-t)\mathbf{x}_2)$.

We know $max(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq max(t\mathbf{x}_1) + max((1-t)\mathbf{x}_2) = t\ max(\mathbf{x}_1) + (1-t)\ max(\mathbf{x}_2)$. Similarly we can derive: $min(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \geq t\ min(\mathbf{x}_1) + (1-t)\ min(\mathbf{x}_2)$.

Therefore, $g(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq (t\ max(\mathbf{x}_1) + (1-t)\ max(\mathbf{x}_2)) - (t\ min(\mathbf{x}_1) + (1-t)\ min(\mathbf{x}_2)) = tg(\mathbf{x}_1) + (1-t)g(\mathbf{x}_2)$. □

3.3 Analytical Solution Intuition

The relaxations introduced before lead to an analytical solution without requiring iterative solvers. Observe that the objective function in Equ. 8c minimizes the difference between the maximum and minimum values along the Blue axis. To achieve that, the intuition is that we must move the colors closer to each other along the Blue axis while making sure the adjusted colors stay within the respective discriminative ellipsoids.

Exactly how to move the colors falls into two cases. Fig. 6 illustrates the two cases using two examples. Without losing generality, we choose to optimize along the Blue axis in these examples (the case along the Red axis is in principle the same), and we plot the projection of the ellipsoids onto the B-G plane for better visualization.

In the first case (Fig. 6a), there is no single plane that cuts across all ellipsoids. This is because the Lowest of the Highest points of all ellipsoids (LH) is lower than the Highest of the Lowest points of all ellipsoids (LH). The optimal strategy is to move all the colors higher than HL toward HL and move all the colors lower than LH toward LH. The movement is necessarily executed along the *extrema vector*, which is the vector that connects the highest and the lowest point of an ellipsoid. After the adjustment, the Blue channels across all the pixels are either HL or LH. That is, the maximum Δ along the Blue axis is now HL − LH, which is the smallest gap we can get the Blue channels to be without going outside the ellipsoid boundaries.

In the second case (Fig. 6b), there is a common plane (*P*) that cuts across all four ellipsoids. In fact, there are infinitely many such planes, because LH is higher HL; thus, any plane between LH and HL will cut across all ellipsoids. In this case, we can simply pick any such plane and move all the colors to that plane. For the simplicity of implementation, we choose the average of the LH and the HL planes as the common plane and move colors along the extrema vectors. In this way, the Blue channel value is exactly the same for all pixels, requiring no Δ bit for the Blue channel.

3.4 Overall Compression Algorithm

We illustrate how our color adjustment algorithm fits in the overall rendering and compression pipeline in Fig. 7. Our adjustment algorithm takes as inputs a tile of pixels (each with three channels) and the parameters of their corresponding discrimination ellipsoids. The algorithm generates the perceptually-adjusted pixel tile as the output. We apply the same color adjustment strategy along both the Blue and the Red axis for each tile, and pick the better one in the end.

It is worth noting that our algorithm does not directly perform compression in itself; it simply adjusts pixel colors so that the (numerically lossless) BD encoding later can achieve higher compression rate. Specifically, the adjusted pixel tile will be first transformed from the linear RGB to the sRGB space, which then goes through the usual BD compression.

Ellipsoid Transformation. The first step in our algorithm is to transform the discrimination ellipsoids from the DKL space to the linear RGB space, which is where color

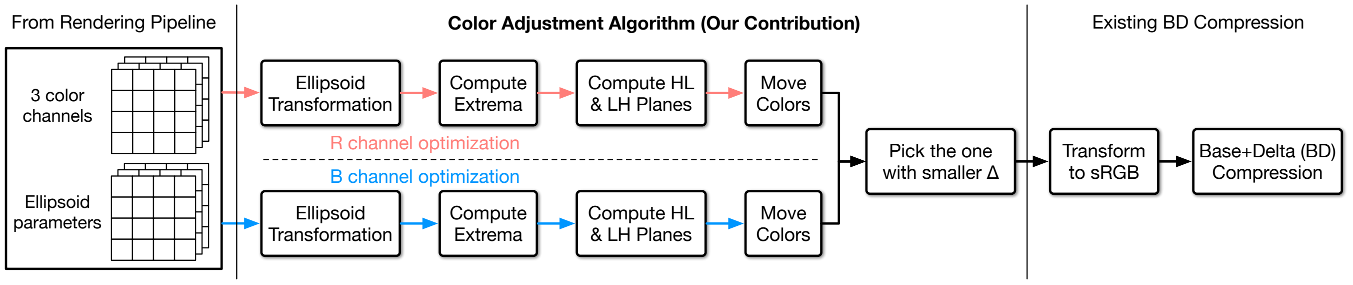


Fig. 7. Overview of our algorithm and how it fits in existing rendering and compression pipeline. Our algorithm takes a tile of pixels and their corresponding discrimination ellipsoid parameters, and generate an adjusted pixel tile, which then goes through existing BD encoding.

adjustment takes place (Sec. 3.3). While ellipsoids are axis-aligned in the DKL color space [22], they will not be axis-aligned after the linear transformation from the DKL to the RGB color space. Therefore, an ellipsoid in the linear RGB space has to take the form of a general quadric surface:

$$Ax^2 + By^2 + Cz^2 + Dx + Ey + Fz + Gxy + Hyz + Izx + 1 = 0 \quad (9)$$

Transforming an axis-aligned ellipsoid in the DKL space to an ellipsoid in the linear RGB amounts to the following matrix multiplication:

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \end{bmatrix} = \left[\begin{array}{c|c} (T \odot T)^\top & \mathbf{0} \\ \hline \mathbf{0} & T \\ \hline \begin{bmatrix} 2T_{00}T_{01} & 2T_{10}T_{11} & 2T_{20}T_{21} \\ 2T_{01}T_{02} & 2T_{11}T_{12} & 2T_{21}T_{22} \\ 2T_{00}T_{02} & 2T_{10}T_{12} & 2T_{20}T_{22} \end{bmatrix} & \mathbf{0} \end{array}\right] \times \begin{bmatrix} 1/a^2t \\ 1/b^2t \\ 1/c^2t \\ -2\kappa_1/a^2t \\ -2\kappa_2/b^2t \\ -2\kappa_3/c^2t \end{bmatrix},$$

$$t = 1 - (\frac{\kappa_1^2}{a^2} + \frac{\kappa_2^2}{b^2} + \frac{\kappa_3^2}{c^2}) \quad (10)$$

where $T = \begin{bmatrix} T_{00} & T_{01} & T_{02} \\ T_{10} & T_{11} & T_{12} \\ T_{20} & T_{21} & T_{22} \end{bmatrix}$ is the constant $\mathrm{M_{RGB2DKL}}$ matrix in Sec. 2.1, $\odot$ is element-wise product, $(\kappa_1, \kappa_2, \kappa_3)$ is the color in DKL space, and (a, b, c) are the semi-axis lengths of κ's discrimination ellipsoids. The derivation uses basic linear transformations and is omitted here due to space constraints.

Color Adjustment. Once we have the ellipsoids in the linear RGB space, we can perform color adjustment, which, as illustrated in Fig. 6 and described in Sec. 3.3, is done in three steps: 1) compute the extrema, i.e., the highest and the lowest point, of each ellipsoid; 2) compute LH and HL based on the extrema of all ellipsoids; 3) compare LH and HL and move colors along extrema vectors accordingly. Step 2 and 3 are relatively straightforward, so here we focus on the mathematical details of Step 1.

Extrema along the Blue axis can be computed by taking the partial derivatives of the ellipsoid equation along the Red and Green axes:

$$\frac{dz}{dx} = 2Ax + Gy + Iz + D = 0 \quad (11a)$$

$$\frac{dz}{dy} = Gx + 2By + Hz + E = 0 \quad (11b)$$

These partial derivatives give us two planes, the intersection of which is a vector **v** that connects the two extrema. The extreme vector **v** is calculated by taking the cross product of the normal vectors of the two planes:

$$\mathbf{v} = (2A, G, I) \times (G, 2B, H) \quad (12)$$

The two extrema points H and L are then calculated by finding the intersection of **v** and the ellipsoid:

$$\mathbf{x} := (x_1, x_2, x_3) = \mathrm{M_{RGB2DKL}} \times \mathbf{v}^T \quad (13a)$$

$$t = 1/\sqrt{\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} + \frac{x_3^2}{c^2}} \quad (13b)$$

$$H = \mathrm{M_{RGB2DKL}^{-1}} \times (\kappa_1 + x_1t, \kappa_2 + x_2t, \kappa_3 + x_3t)^T$$
$$L = \mathrm{M_{RGB2DKL}^{-1}} \times (\kappa_1 - x_1t, \kappa_2 - x_2t, \kappa_3 - x_3t)^T \quad (13c)$$

where κ is the pixel color in the DKL space, (a, b, c) are DKL ellipsoid parameters, and $\mathrm{M_{RGB2DKL}}$ is the RGB to DKL transformation matrix (Sec. 2.1). We omit the derivation details due to space constraints, but the derivation amounts to a simple application of line-ellipsoid intersection and linear transformations between RGB and DKL space.

Remarks on Decoding. One desired byproduct of our algorithm is that it requires *no* change to the existing frame-buffer decoding scheme — our color adjustment algorithm simply changes the input to BD. During decoding (e.g., by the display controller), the existing BD decoder will construct the sRGB values from the BD-encoded data, which are then sent to the display. The exact BD encoding format varies across implementations and is not our focus. We assume the encoding format described in Zhang et al. [76].

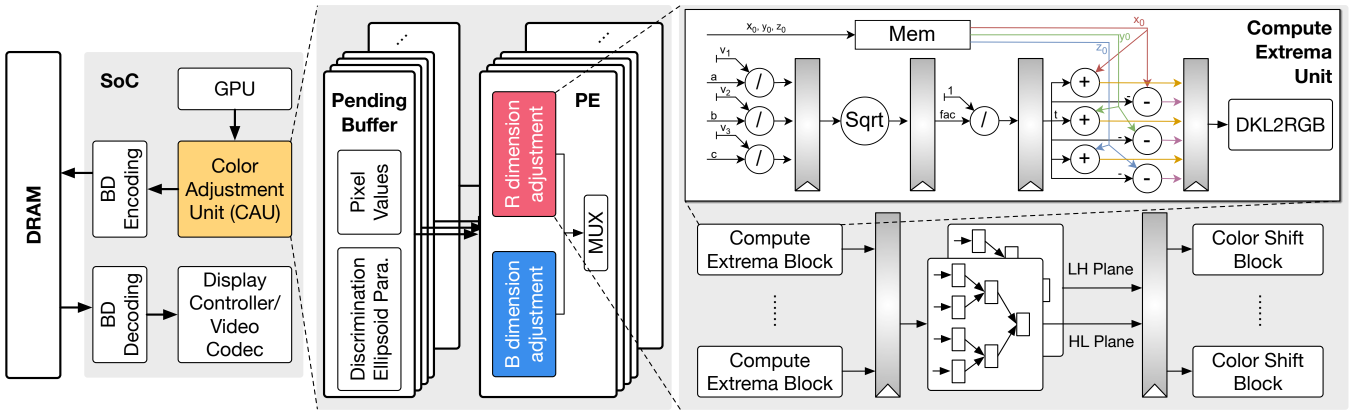


Fig. 8. Illustration of the hardware support, which we dub Color Adjustment Unit (CAU) for our image encoding and how CAU interfaces with the rest of the SoC. Internally, the CAU uses an array of PEs, each of which adjust colors for one tile of pixels. CAU is fully pipelined to accept a new tile every cycle from the Pending Buffer, which receives the rendered pixels and their discrimination ellipsoids from the GPUs.

4 Hardware Architecture

The analytical compression algorithm, while avoiding iterative solvers, is still compute intensive and slow to execute in real-time. We implement it as a GPU shader executing on the Adreno 650 GPU in Oculus Quest 2, a widely used mobile VR headset. The compression algorithm runs in a mere 2 FPS. This section describes a lightweight hardware design that accelerates the compression algorithm. Sec. 4.1 describes how our custom hardware fits into the overall system and Sec. 4.2 describes the hardware details.

4.1 Hardware Overview

Fig. 8 provides an overview of our architectural extension, dubbed the Color Adjustment Unit (CAU), and how CAU fits into existing mobile SoCs. The CAU executes the pixel adjustment algorithm described in Sec. 3. The CAU reads its input from an on-chip buffer, which stores the pixels and the discrimination ellipsoid parameters generated by the GPU. Following prior work [22], we assume that the GPU is responsible for generating the per-pixel discrimination ellipsoids. The generation algorithm is a lightweight RBF network (Sec. 2.1). In our measurement, the ellipsoid generation algorithm on Oculus Quest 2 runs at the maximum display refresh rate (72 FPS) while consuming less than 1 mW measured using Oculus' OVR Metrics Tool [5].

The output of the CAU enters the existing BD framebuffer encoder, which writes the encoded data to the DRAM. Any frame read out from the DRAM, e,g., by the Displayer Controller IP block when sending the frame to the display, will enter the BD decoder, which reconstructs the sRGB pixels. The figure provides a visual confirmation that our algorithm 1) works on top of, rather than replaces, BD encoding, and 2) does not change the decoding architecture.

4.2 Color Adjustment Unit

Internally, the CAU consists of an array of Processing Elements (PEs), each of which is designed to adjust colors for *one tile* of pixels, which in our current design is assumed to be 4×4. Each PE interfaces with a dedicated Pending Buffer, which holds all the information of the pixel tiles generated from the GPU. Having more PEs will allows the system to compressing multiples tiles simultaneously.

Pipelining. The PE is fully pipelined to accept a new tile every cycle. Fig. 8 illustrates the detailed architecture, which has three main phases, each of which is internally pipelined. The first phase computes the extrema. The next phases use reduction trees to calculate HL and LH from the extrema. The final phase move the colors along the extrema vector.

Compute Extrema Blocks. This component calculates the extrema of all the pixels in a tile, which is naturally parallelizable across pixel and, thus, has multiple parallel units, each of which is responsible for one pixel. The top-right box in Fig. 8 illustrates the microarchitecture. This is the most compute intensive block in the CAU, since it involves multiple divisions and square root operations. The division and square root hardware implements Equ. 13b, and the adder and subtractor circuit implements Equ. 13c. The DKL-RGB transformations in Equ. 13c and Equ. 13a are implemented through matrix vector multiplication executed on a 3×3 MAC array.

Compute Planes Blocks. The extrema calculated before enters this unit, which finds the channel value for the HL plane (maximum of the minima) and LH (minimum of the maxima) plane. We implement this stage using two reduction (comparator) trees to generate both planes simultaneously.

Color Shift Blocks. This block takes the original color values and the two planes as input and outputs the modified color values. This phase is control-flow heavy, as it involves multiple condition checks, e.g., testing the relationship between a point and a plane. A custom datapath in CAU avoids much of the inefficiencies surrounding control flows that are detrimental to GPU performance. This hardware is a relatively straightforward mapping from the algorithm.

Pending Buffer. The Pending Buffers store intermediate pixels and their discrimination ellipsoids from the GPU before they are consumed by the CAU. Each buffer is interfaced with a dedicated PE and, thus, contains the data for all the pixel tiles to be consumed by the PE.

The buffers must be properly sized so as to not stall or starve the CAU pipeline. In order to be independent of the exact GPU microarchitecture details, we make a conservative estimation of the buffer size. In particular, we allocate enough space in the buffer such that it can hold all the pixels generated by the GPU in each CAU cycle *even if* the GPU is fully utilized, in which case each shader core in the GPU generates 1 pixel/GPU cycle. Note that the GPU and CAU cycle times need not be the same. The number of PEs in a CAU must be properly decided so as to not stall either the GPU nor the CAU, as we will discuss in Sec. 6.1.

5 Experimental Methodology

5.1 Setup

Hardware. We implement our encoder and decoder units in SystemVerilog and use an EDA flow consisting of Synopsys and Cadence tools with the TSMC 7 nm FinFET technology to obtain latency and area. We use Synopsys DesignWare library for a variety of RTL implementations such as the pipelined divider. Power is estimated using Synopsys PrimeTimePX with fully annotated switching activity.

The DRAM energy is calculated using Micron's System Power Calculators [4], assuming a typical 8 Gb, 32-bit LPDDR4. On average, the DRAM access energy per pixel is estimated to be 3,477 pJ/pixel, matching prior work [33, 34].

Dataset and Software. We evaluate our compression algorithm with 6 different VR scenes used in VR color perception studies [22]. In each scene, each frame is rendered with two sub-frames, one for each eye. All the frames are dynamically rendered (on the GPU) at run time, i.e., the frames are neither loaded from the disk nor streamed over the network. Following the common practice in color perception studies [14, 22], we keep pixels in the central 10° FoV unchanged, and apply the compression algorithm only on the rest (peripheral) pixels.

As discussed in Sec. 3.4, our algorithm works in conjunction with existing BD compression. In this paper, we assume a recent, state-of-the-art, BD algorithm described by Zhang et al. [76], from which we obtain the final compression rate.

5.2 Human Subject Studies

We also evaluate the perceptual quality of our compression algorithm on actual participants. We recruit 11 participants

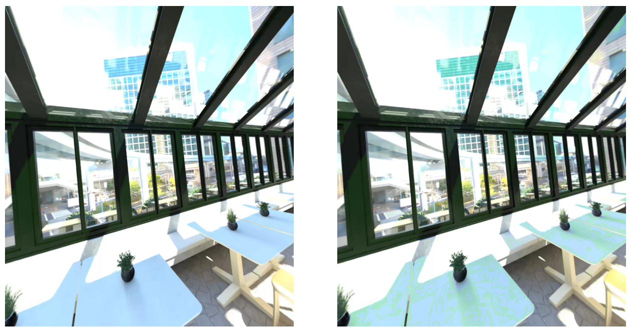

(a) Original frame. **(b)** Color-adjusted frame.

Fig. 9. A pair of images without (left) and with (right) our color adjustment. The two images when viewed on a conventional computer display are visibly different, because the entirety of the images will be in the viewer's foveal vision.

(3 female; ages between 19 and 40). None of the participants were aware of the research, the number of conditions, or the hypothesis before taking the experiments, which were approved by an Internal Review Board.

We face a dilemma in user study: the speed of the compression algorithm implemented as a GPU shader is too slow on today's mobile VR headsets (e.g., 2 FPS on Oculus Quest 2 as discussed in Sec. 4) — the motivation behind our architectural support, but this also means we can not use a mobile VR headset for user study. Our approach is to run the user study on a tethered VR headset, HTC Vive Pro Eye, which is connected to a PC with a powerful Nvidia RTX A2000 GPU, which runs the compression algorithm at 90 FPS, sufficient for user study.

Each participant was shown the six VR scenes (20 seconds each) used in a prior study [22] in random order. To encourage and ensure that the participants actively and freely explored the scene, each participant was asked to perform a scene-specific task, such as counting the number of birds in the scene. At the end of each video, we asked the participant whether they notice any visual artifacts.

In order for participants to isolate potential artifacts introduced by our compression from other irrelevant artifacts (e.g., low resolution, aliasing in rendering), at the beginning of each test we show the participant two images on a computer display, one with and the other without our perceptual compression; see examples in Fig. 9. When participants viewed the images on the computer display, the entire frames were in their foveal vision so the color adjustment was clearly visible. In this way, we make sure the artifacts reported by users resulted from compression. This is a common practice in subjective color perception studies [22]. The user study results should be seen as the *lower bound* of the quality of compression algorithm, because the participants were aware of and thus better identification of the artifacts.

5.3 Baselines

We compare against four baselines:

- NoCom: no compression;
- BD: existing BD compression based on Zhang et al. [76];
- PNG: lossless compression based on the popular Portable Network Graphics (PNG), which is unsuitable for real-time DRAM compression because of its high run-time overhead even with dedicated hardware acceleration [1, 30]. For instance, the commercial IPB-PNG-E FPGA-based IP core compresses an 800×600 image only at a 20 FPS [1].
- SCC: an alternative strategy to exploit color discrimination based on the Set Cover formulation, which we describe next.

SCC uses a look-up table to map each 24-bit sRGB color to a more compact encoding. This can be formulated as a set cover problem [32]: find the smallest subset of sRGB colors C ⊂ sRGB whose discrimination ellipsoids union-ed together cover all the 2^{24} sRGB colors. Each new color is then encoded with only $log_2\lceil|C|\rceil$ bits, where $|\cdot|$ denotes the set cardinality.

The set cover problem is a classic NP-complete problem [32], where the optimal solution requires combinatorial search. We use a common greedy heuristics [13] and construct the mapping tables. The encoding table consumes 30 MB and the decoding table consumes 96 KB, too large for SCC to be used for DRAM traffic compression in mobile SoCs.

6 Evaluation

We first show that the area and power overhead of our compression scheme is negligible while ensuring real-time compression (Sec. 6.1). We then present the benefits of our compression scheme in DRAM traffic reduction and power savings, and analyze the sources of the savings (Sec. 6.2). We then present our human subject studies, which show that our compression scheme introduces little visible artifacts (Sec. 6.3). We present a sensitivity study of the key parameters in our compression scheme (Sec. 6.4). Finally, we discuss how we can accommodate a diverse range of users (Sec. 6.5).

6.1 Area and Power Overhead

Performance. Our algorithm along with the hardware support achieves real-time compression. The CAU operates with a cycle time of about 6 *ns*, which translates to a frequency of about 166.7 MHz. The Adreno 650 GPU used in Oculus Quest 2 operates at a nominal frequency of 441 MHz, which means during each CAU cycle (at most) three pixels are generated by a shader core in the GPU. Given that the Adreno 650 GPU has 512 shader cores, each CAU cycle 512×3 pixels (i.e., 96 tiles) are generated. Therefore, we configure our CAU to have 96 PEs, which are able to process 96 tiles simultaneously, matching peak throughput of the GPU.

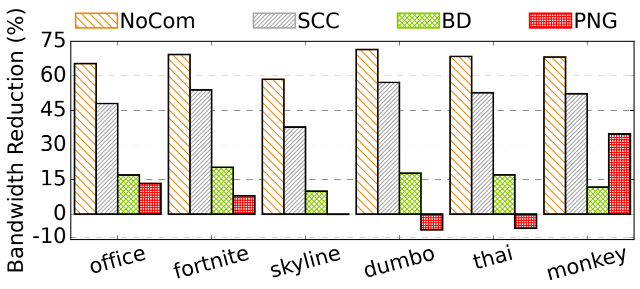


Fig. 10. Bandwidth reduction over baselines.

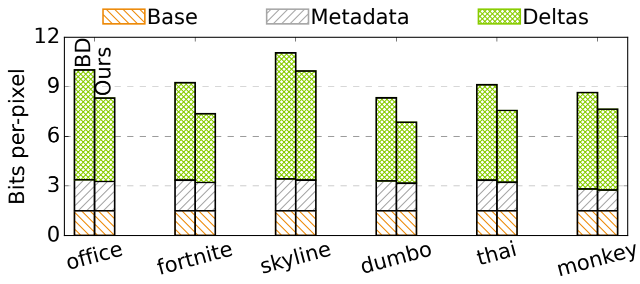


Fig. 11. Distribution of bits per pixel across the three components: base, metadata, and Δ. Left: BD; Right: our algorithm.

Thus, when compressing a 5408×2736 image (the highest rendering resolution on Oculus Quest 2), compression adds a delay of 173.4 μs, negligible in a rendering pipeline that operates at, say, 72 FPS with a frame time budget of 13.9 *ms*.

Area and Power. Our compression hardware extension introduces little area overhead, which consists of that of the Pending Buffers and the PEs. Each PE has an area of 0.022 mm^2, resulting in a total PE size of 2.1 mm^2. Each Pending Buffer holds data for two tiles (double buffering); the total buffer size is 36 KB, resulting in a total area of 0.03 mm^2.

The area overhead is negligible compared to the size of a typical mobile SoC. For instance, the Xavier SoC has an area of 350 mm^2 (12 nm) [6], Qualcomm Snapdragon 865 SoC has a die area of 83.54 mm^2 (7 nm) [8], and Apple A14 SoC has a die area of 88 mm^2 (5 nm) [2]. The power consumption of each PE and its buffer is about 2.1 μW, resulting in a total CAU power consumption of about 201.6 μW, which we faithfully account for in the power saving analyses later.

6.2 Results

Compression Rate. Fig. 10 shows the bandwidth reduction of our algorithm compared to the baselines. Our algorithm achieves a compression rate of 66.9%, 50.3%, and 15.6% over NoCom, SCC, and BD, respectively. Unsurprisingly, the highest gains are against NoCom, which is the original frames and uses 3 Bytes (24 bits) to store each pixel.

SCC (Sec. 5.3) is able to map all the 2^{24} (about 16.8 million) sRGB colors to a small subset of only 32,274 colors.

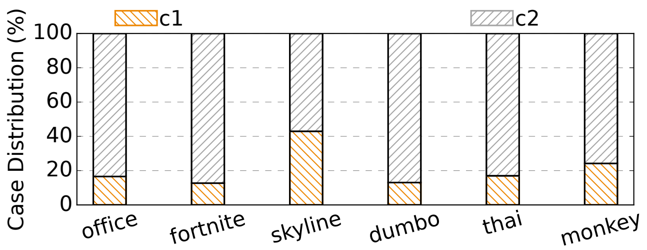


Fig. 12. Distribution of the two cases c1 and c2.

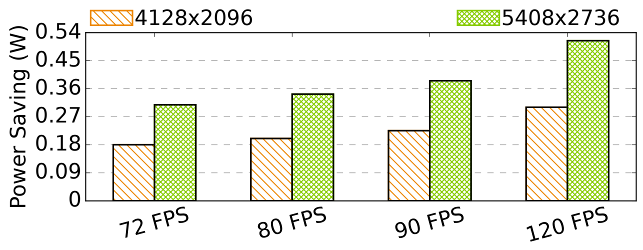


Fig. 13. Power saving over BD under the lowest and highest resolutions and four different frame rates on Oculus Quest 2.

SCC thus uses 15 bits to represent a color, reducing the storage cost compared to the original frames but is still much less efficient than BD, which is the canonical Base+Delta approach to compression DRAM traffic in today's mobile SoCs. Compared to BD, we show 15.6% (up to 20.4%) higher compression rate, because of our ability to exploit human color discrimination to reduce the magnitudes of Δs.

We get the least improvement over PNG. In two scenes, PNG actually has a higher compression rate. This matches prior results on BD [76] and is not surprising — to get a high compression rate PNG is computationally intensive and is meant to be used offline; see discussion in Sec. 5.3.

Understanding Results. Our improvement over BD comes from the fact that we require fewer bits to store the Δs. Fig. 11 shows the average number of bits per pixel required to store the base, metadata, and Δs in a tile. We compare the statistics between BD (left bars) and our scheme (right bars). It is clear that the space reduction comes from reducing the number of bits required to store the Δs.

To dissect how our scheme reduces the magnitude of Δs, Fig. 12 shows the distribution of tiles across the two cases in Fig. 6: HL > LH (c1) and HL < LH (c2). We observe that c2 is the more common case: 78.92% tiles result in this case. In c2, there exists a common plane where all the color values can collapse to. We can reduce the Δ to 0 in these tiles, essentially eliminating the need to store Δ.

Power Reduction. We evaluate the power reduction under different resolutions and frame rates available on Oculus Quest 2. Fig. 13 shows the power savings under each combination over BD. Across all configurations, we reduce the power consumption by 307.2 mW on average. The power

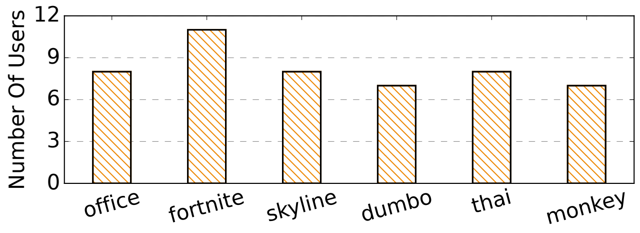


Fig. 14. Number of participants (out of 11) who did *not* notice any artifacts in each scene in our user study.

saving is a combination of reducing the DRAM traffic and the power overhead of the CAU encoding (201.6 μW).

Even on the lowest resolution and frame rate combination on Oculus Quest 2, we reduce the power consumption by 180.3 mW, which translates to about 29.9% of the total power measured (using Oculus' OVR Metrics Tool [5]) when rendering without compression. Under the highest resolution and frame rate combination, the power saving increases to 514.2 mW. As resolution and frame rate will likely increase in future VR devices, the power benefits of our compression scheme will only increase.

6.3 User Studies and Analyses

Fig. 14 shows the number of participants who did *not* notice any artifact in each scene. On average, 2.8 participants (standard deviation 1.5) out of 11 total participants observe artifacts. This percentage is on par with prior color perception studies [14, 22]. We further interviewed the participants and identified three reasons why certain participants notice artifacts, all of which were orthogonal to the fundamental idea of this paper and actually point to optimization opportunities in *other* parts of the system, which, when exploited, can be readily integrated into our work.

One participant who noticed subtle artifacts in three out of the six scenes was a visual artist with "color-sensitive eyes." Observer variation is a known phenomenon in vision science since the early days of color science research [26, 67, 72]. Given that color discrimination models in the literature all target the average case in the population, the results indicate that customizing/calibrating the model for individual users is likely a promising approach to reduce the artifact.

Another set of participants noticed artifacts only during rapid eye/head movement but not with a steady pose. This is likely attributed to external factors such as rendering lag or slow gaze detection, which is independent of our algorithm.

Finally, we found that no participant noticed any artifact in the fortnite scene, which is a bright scene with a large amount of green. Since our compression algorithm generally yields green-hue shifts (see examples in Fig. 9), artifacts are less noticeable in scenes that are green to begin with. In contrast, dumbo and monkey, both dark scenes, have the most noticeable artifacts. The results suggest, to the

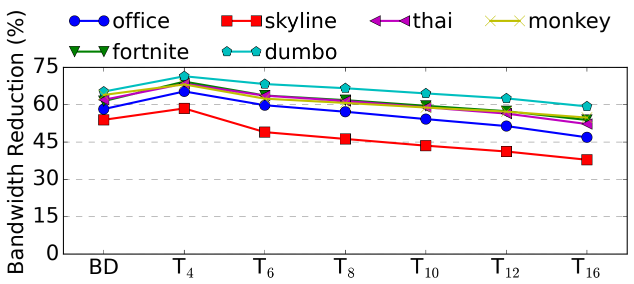


Fig. 15. Bandwidth reduction over NoCom under BD and our scheme with different tile sizes denoted by T_n, where n is the tile size.

vision science community, the need for improving the color discrimination models in low-luminance conditions.

Objective Image Quality. To show that subjective experience, which is the focus of our work, is *not* equivalent to objective quality, we evaluate the Peak-Signal-to-Noise-Ratio (PSNR), a common objective quality metric, of all the compressed images. On average, the PSNR of the compressed videos is 46.0 dB (standard deviation 19.5); all but two scenes have a PSNR below 37. A PSNR value in this range usually indicates noticeable visual artifacts [12], which is confirmed by our participants when they view the compressed images on a conventional display. This result accentuates the crux of our work: use human color perception in VR to guide a numerically lossy scheme (hence low PSNR) for higher compression rate with little subjective quality degradation.

6.4 Sensitivity Studies

Our evaluation so far assumes a tile size of 4×4. We also evaluate our compression algorithm across different tile sizes; the results are shown in Fig. 15 along with BD. We observe that the compression rate drops once the tile size increases beyond 4×4 and can be worse than BD when the tile size is larger than 8×8.

The trend is the result of two opposing effects. On one hand, as we increase the tile size we can amortize the cost of storing the base pixels. On the other hand, larger tiles also present less opportunity to bringing pixels together, because we have to accommodate the worst case/largest difference between two pixels in a tile (Sec. 3.1).

6.5 Discussions

To accommodate individual color perception in actual system deployments, one can perform a per-user color calibration procedure to build a per-user ellipsoid model. Such a procedure is laid out in prior work [22], and is readily doable. Such user-specific calibrations are routinely done when a user first uses an AR/VR product, e.g., adjusting the pair of displays to accommodate different inter-pupil distances among individuals. When a per-user ellipsoid model is available, our fundamental idea readily applies.

It is worth noting that we can, if need be, easily turn off our compression algorithm, which is intentionally designed as a plug-and-play stage between normal GPU rendering and existing BD compression (see Fig. 7). One scenario where one might want to turn off our compression is when a user has color vision deficiency (CVD). The color discrimination model that underlies our compression algorithm does not consider individuals with CVD. When such models for CVD become available, our fundamental idea readily applies.

7 Related Work

Perception-Aware Rendering. A host of recent work has focused on leveraging human perception to optimize AR/VR systems. Weier et al. provide a relatively recent survey [65]. The most studied approach is foveated rendering, which reduces rendering resolution in the visual periphery [25, 46, 58, 59, 64]. Foveated rendering has been theoretically studied to reduce data transmission traffic in cloud rendering [31, 35], but the decoding (reconstruction) cost is prohibitively high (e.g., need a complicated DNN). Our approach is orthogonal to foveated rendering in that we focus on adjusting colors rather than the spatial frequency, and works directly on top of the existing (BD-based) framebuffer compression framework without adding decoding cost.

Color Perception in Systems Optimizations. Color perception is most often leveraged to reduce display energy. To our best knowledge, this is the first paper that leverages color perception to reduce data communication energy.

Dong et al. [20], Crayon [57], Dong and Zhong [21] all leverage the human color discrimination to reduce OLED power, which is known to strongly correlate with color. Duinkharjav et al. [22] extend this approach to VR by quantifying the eccentricity dependent color discrimination. Recent work by Dash and Hu [18] builds an accurate color-vs-display power model. None focused on reducing data traffic. Shye et al. [55] and Yan et al. [73] leverage dark adaptation to reduce display power. Dark adaptation will likely weaken the color discrimination even more, potentially further improving the compression rate — an interesting future direction.

Data Traffic Optimizations in VR. Data traffic reduction in VR has mostly been studied in the context of client-cloud collaborative rendering, i.e., reducing wireless transmission traffic. The pioneering Furion [37] system and later developments and variants such as Coterie [42] and Q-VR [71] cleverly decide what to rendering locally vs. remotely. For instance, one could offload background/far objects rendering to the server and render foreground/near object interactions locally. EVR [38, 60] predicts user FoV trajectory and pre-renders VR videos in the cloud. Our proposal is orthogonal to the client-remote collaborative rendering, in that we focus on reducing DRAM traffic occurred within a local device.

Zhang et al. [76] describe a BD design in encoding framebuffer traffic. We directly compare against this approach and show up to 20% bandwidth savings. Zhang et al. [75] propose a content cache that exploits value equality in video decoding, which does not apply to encoding where strict equality is rare. Rhythmic Pixel Regions [34] drops pixel tiles to reduce DRAM traffic in a machine vision pipeline, whereas our focus is human visual perception in VR.

Any compression algorithm, ours included, exploits data similarities. Recent work leverages data similarities to speedup rendering [66, 74, 77, 78] by eliding redundant computations (that compute same/similar data). These methods, however, do not reduce data traffic, which we do.

General Memory Compression. Exploiting value similarities to compress data traffic is a long-standing technique in architecture [47, 48]. Recent work in approximate computing extends compression to tasks that can tolerate slight data infidelity such as image processing [43, 53] and texture mapping in rendering [61, 70]. In comparison, this paper performs a principled "approximate compression" by 1) using a rigorous human perception model derived from psychophysical experiments and 2) formulating compression as a constraint optimization with an optimal solution (under necessary relaxations). Finally, we specifically target VR and, thus, exploit the eccentricity dependency that is unconcerned with before.

8 Conclusion

Aggressively lossy compression in the numerical domain can achieve significant data traffic reduction with little perceptual quality loss in VR. The key is to leverage human color discrimination (in)ability to bring pixels more similar to each other. The resulting images, thus, permit more aggressive compression over the classic Base+Delta scheme to reduce DRAM traffic in a mobile SoC. We show that our compression algorithm has an analytical form, which, when accelerated by a dedicated hardware, can achieve real-time compression. Future VR systems design must actively integrate human perception into the optimization loop.

References

[1] 24-Bit 20-FPS PNG Encoder. https://ipblog.files.wordpress.com/2017/09/ipb-png-e-pb.pdf.
[2] Apple's A14 SoC Under the Microscope: Die Size & Transistor Density Revealed. https://www.tomshardware.com/news/apple-a14-bionic-revealed.
[3] Arm Frame Buffer Compressions (AFBC). https://www.arm.com/technologies/graphics-technologies/arm-frame-buffer-compression.
[4] Micron System Power Calculators. https://www.micron.com/support/tools-and-utilities/power-calc.
[5] OVR Metrics Tool. https://developer.oculus.com/downloads/package/ovr-metrics-tool/.
[6] Tegra Xavier. https://en.wikichip.org/wiki/nvidia/tegra/xavier.
[7] The Next Frontier For Healthcare: Augmented Reality, Virtual Reality, And The Metaverse. https://www.forbes.com/sites/saibala/2021/11/29/the-next-frontier-for-healthcare-augmented-reality-virtual-reality-and-the-metaverse/?sh=468f6be22894.
[8] Xiaomi Mi 10 Teardown Analysis. https://www.techinsights.com/blog/xiaomi-mi-10-teardown-analysis.
[9] Murat Akçayır and Gökçe Akçayır. Advantages and challenges associated with augmented reality for education: A systematic review of the literature. *Educational Research Review*, 20:1–11, 2017.
[10] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. Latency requirements for foveated rendering in virtual reality. *ACM Transactions on Applied Perception (TAP)*, 14(4):1–13, 2017.
[11] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
[12] Yang Cao, Tao Jiang, Xu Chen, and Junshan Zhang. Social-aware video multicast based on device-to-device communications. *IEEE Transactions on Mobile Computing*, 15(6):1528–1539, 2015.
[13] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.
[14] Michael A. Cohen, Thomas L. Botch, and Caroline E. Robertson. The limits of color awareness during active, real-world vision. *Proceedings of the National Academy of Sciences*, 117(24):13821–13827, 2020.
[15] Christine A Curcio and Kimberly A Allen. Topography of ganglion cells in human retina. *Journal of comparative Neurology*, 300(1):5–25, 1990.
[16] Christine A Curcio, Kenneth R Sloan, Robert E Kalina, and Anita E Hendrickson. Human photoreceptor topography. *Journal of comparative neurology*, 292(4):497–523, 1990.
[17] Dennis M Dacey. The mosaic of midget ganglion cells in the human retina. *Journal of Neuroscience*, 13(12):5334–5355, 1993.
[18] Pranab Dash and Y Charlie Hu. How much battery does dark mode save? an accurate oled display power profiler for modern smartphones. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 323–335, 2021.
[19] Andrew M Derrington, John Krauskopf, and Peter Lennie. Chromatic mechanisms in lateral geniculate nucleus of macaque. *The Journal of physiology*, 357(1):241–265, 1984.
[20] Mian Dong, Yung-Seok Kevin Choi, and Lin Zhong. Power modeling of graphical user interfaces on oled displays. In *Proceedings of the 46th Annual Design Automation Conference*, pages 652–657, 2009.
[21] Mian Dong and Lin Zhong. Chameleon: A color-adaptive web browser for mobile oled displays. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 85–98, 2011.
[22] Budmonde Duinkharjav, Kenneth Chen, Abhishek Tyagi, Jiayi He, Yuhao Zhu, and Qi Sun. Color-perception-guided display power reduction for virtual reality. *ACM Transactions on Graphics (TOG)*, 41(6):1–16, 2022.
[23] Michele Fiorentino, Raffaele de Amicis, Giuseppe Monno, and Andre Stork. Spacedesign: A mixed reality workspace for aesthetic industrial design. In *Proceedings. International Symposium on Mixed and Augmented Reality*, pages 86–318. IEEE, 2002.
[24] Edward J Giorgianni, Thomas E Madden, and Kevin E Spaulding. Color management for digital imaging systems. In *Digital color imaging handbook*, pages 239–268. CRC Press, 2017.
[25] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. Foveated 3d graphics. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.
[26] John Guild. The colorimetric properties of the spectrum. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 230(681-693):149–187, 1931.
[27] Jawad Haj-Yahya, Jisung Park, Rahul Bera, Juan Gómez Luna, Efraim Rotem, Taha Shahroodi, Jeremie Kim, and Onur Mutlu. Burstlink: Techniques for energy-efficient video display for conventional and virtual reality systems. In *MICRO-54: 54th Annual IEEE/ACM International*

Symposium on Microarchitecture, pages 155–169, 2021.

[28] Xu Han, Ying Chen, Qinna Feng, and Heng Luo. Augmented reality in professional training: A review of the literature from 2001 to 2020. *Applied Sciences*, 12(3):1024, 2022.

[29] Thorsten Hansen, Martin Giesel, and Karl R Gegenfurtner. Chromatic discrimination of natural objects. *Journal of Vision*, 8(1):2–2, 2008.

[30] Shizhen Huang and Tianyi Zheng. Hardware design for accelerating png decode. In *2008 IEEE International Conference on Electron Devices and Solid-State Circuits*, pages 1–4. IEEE, 2008.

[31] Anton S Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. Deepfovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019.

[32] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010.

[33] Venkatesh Kodukula, Saad Katrawala, Britton Jones, Carole-Jean Wu, and Robert LiKamWa. Dynamic temperature management of near-sensor processing for energy-efficient high-fidelity imaging. *Sensors*, 21(3):926, 2021.

[34] Venkatesh Kodukula, Alexander Shearer, Van Nguyen, Srinivas Lingutla, Yifei Liu, and Robert LiKamWa. Rhythmic pixel regions: multi-resolution visual sensing system towards high-precision visual computing at low power. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 573–586, 2021.

[35] Brooke Krajancich, Petr Kellnhofer, and Gordon Wetzstein. A perceptual model for eccentricity-dependent spatio-temporal flicker fusion and its applications to foveated graphics. *ACM Trans. Graph.*, 40, 2021.

[36] John Krauskopf and Gegenfurtner Karl. Color discrimination and adaptation. *Vision research*, 32(11):2165–2175, 1992.

[37] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 409–421, 2017.

[38] Yue Leng, Chi-Chun Chen, Qiuyue Sun, Jian Huang, and Yuhao Zhu. Energy-efficient video processing for virtual reality. In *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[39] Chiao Liu, Andrew Berkovich, Song Chen, Hans Reyserhove, Syed Shakib Sarwar, and Tsung-Hsun Tsai. Intelligent vision systems–bringing human-machine interface to ar/vr. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 10–5. IEEE, 2019.

[40] Chiao Liu, Song Chen, Tsung-Hsun Tsai, Barbara De Salvo, and Jorge Gomez. Augmented reality-the next frontier of image sensors and compute systems. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 426–428. IEEE, 2022.

[41] David L MacAdam. Visual sensitivities to color differences in daylight. *Josa*, 32(5):247–274, 1942.

[42] Jiayi Meng, Sibendu Paul, and Y Charlie Hu. Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 923–937, 2020.

[43] Joshua San Miguel, Jorge Albericio, Andreas Moshovos, and Natalie Enright Jerger. Doppelgänger: a cache for approximate computing. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 50–61, 2015.

[44] Ohan Oda, Carmine Elvezio, Mengu Sukan, Steven Feiner, and Barbara Tversky. Virtual replicas for remote assistance in virtual and augmented reality. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 405–415, 2015.

[45] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Transactions on Graphics (TOG)*, 35(6):1–12, 2016.

[46] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph.*, 35(6), November 2016.

[47] Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Linearly compressed pages: A low-complexity, low-latency main memory compression framework. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–184, 2013.

[48] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Base-delta-immediate compression: Practical data compression for on-chip caches. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 377–388, 2012.

[49] Thammathip Piumsomboon, Gun A Lee, Jonathon D Hart, Barrett Ens, Robert W Lindeman, Bruce H Thomas, and Mark Billinghurst. Mini-me: An adaptive avatar for mixed reality remote collaboration. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–13, 2018.

[50] Charles Poynton. *Digital video and HD: Algorithms and Interfaces*. Elsevier, 2012.

[51] Iain E Richardson. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[52] RW Rodieck, KF Binmoeller, and J Dineen. Parasol and midget ganglion cells of the human retina. *Journal of Comparative Neurology*, 233(1):115–132, 1985.

[53] Joshua San Miguel, Jorge Albericio, Natalie Enright Jerger, and Aamer Jaleel. The bunker cache for spatio-value approximation. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[54] Lindsay T Sharpe, Andrew Stockman, Wolfgang Jagla, and Herbert Jägle. A luminous efficiency function, v*(λ), for daylight adaptation. *Journal of vision*, 5(11):3–3, 2005.

[55] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*, pages 168–178, 2009.

[56] Hongxin Song, Toco Yuen Ping Chui, Zhangyi Zhong, Ann E Elsner, and Stephen A Burns. Variation of cone photoreceptor packing density with retinal eccentricity and age. *Investigative ophthalmology & visual science*, 52(10):7376–7384, 2011.

[57] Phillip Stanley-Marbell, Virginia Estellers, and Martin Rinard. Crayon: Saving power through shape and color approximation on next-generation displays. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–17, 2016.

[58] Qi Sun, Fu-Chung Huang, Joohwan Kim, Li-Yi Wei, David Luebke, and Arie Kaufman. Perceptually-guided foveation for light field displays. *ACM Trans. Graph.*, 36(6), November 2017.

[59] Qi Sun, Fu-Chung Huang, Li-Yi Wei, David Luebke, Arie Kaufman, and Joohwan Kim. Eccentricity effects on blur and depth perception. *Optics express*, 28(5):6734–6739, 2020.

[60] Qiuyue Sun, Amir Taherin, Yawo Siatitse, and Yuhao Zhu. Energy-efficient 360-degree video rendering on fpga via algorithm-architecture co-design. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 97–103, 2020.

[61] Mark Sutherland, Joshua San Miguel, and Natalie Enright Jerger. Texture cache approximation on gpus. In *Workshop on approximate computing across the stack*, page 3, 2015.

[62] Tom Forsyth. The sRGB Learning Curve. https://medium.com/@tomforsyth/the-srgb-learning-curve-773b7f68cf7a.

[63] Gregory K Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

[64] David R Walton, Rafael Kuffner Dos Anjos, Sebastian Friston, David Swapp, Kaan Akşit, Anthony Steed, and Tobias Ritschel. Beyond blur: Real-time ventral metamers for foveated rendering. *ACM Transactions on Graphics*, 40(4):1–14, 2021.

[65] Martin Weier, Michael Stengel, Thorsten Roth, Piotr Didyk, Elmar Eisemann, Martin Eisemann, Steve Grogorick, André Hinkenjann, Ernst Kruijff, Marcus Magnor, et al. Perception-driven accelerated rendering. In *Computer Graphics Forum*, volume 36, pages 611–643. Wiley Online Library, 2017.

[66] Yu Wen, Chenhao Xie, Shuaiwen Leon Song, and Xin Fu. Post0-vr: Enabling universal realistic rendering for modern vr via exploiting architectural similarity and data sharing. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 390–402. IEEE, 2023.

[67] William David Wright. A re-determination of the trichromatic coefficients of the spectral colours. *Transactions of the Optical Society*, 30(4):141, 1929.

[68] William David Wright and FHG Pitt. Hue-discrimination in normal colour-vision. *Proceedings of the Physical Society*, 46(3):459, 1934.

[69] Günther Wyszecki and Walter Stanley Stiles. *Color science: concepts and methods, quantitative data and formulae*, volume 40. John wiley & sons, 2000.

[70] Chenhao Xie, Xin Fu, and Shuaiwen Song. Perception-oriented 3d rendering approximation for modern graphics processors. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 362–374. IEEE, 2018.

[71] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael Taylor, and Shuaiwen Leon Song. Q-vr: system-level design for future mobile collaborative virtual reality. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 587–599, 2021.

[72] Hao Xie, Susan P Farnand, and Michael J Murdoch. Observer metamerism in commercial displays. *JOSA A*, 37(4):A61–A69, 2020.

[73] Zhisheng Yan, Chen Song, Feng Lin, and Wenyao Xu. Exploring eye adaptation in head-mounted display for energy efficient smartphone virtual reality. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, pages 13–18, 2018.

[74] Ziyu Ying, Shulin Zhao, Haibo Zhang, Cyan Subhra Mishra, Sandeepa Bhuyan, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Exploiting frame similarity for efficient inference on edge devices. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 1073–1084. IEEE, 2022.

[75] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T Kandemir, Ravi Iyer, and Chita R Das. Race-to-sleep+ content caching+ display caching: A recipe for energy-efficient video streaming on handhelds. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 517–531, 2017.

[76] Haibo Zhang, Shulin Zhao, Ashutosh Pattnaik, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Distilling the essence of raw video to reduce memory usage and energy at edge devices. In *Proceedings of the 52nd Annual IEEE/ACM international symposium on microarchitecture*, pages 657–669, 2019.

[77] Shulin Zhao, Haibo Zhang, Sandeepa Bhuyan, Cyan Subhra Mishra, Ziyu Ying, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Déja view: Spatio-temporal compute reuse for ‘energy-efficient 360 vr video streaming. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 241–253. IEEE, 2020.

[78] Shulin Zhao, Haibo Zhang, Cyan Subhra Mishra, Sandeepa Bhuyan, Ziyu Ying, Mahmut Taylan Kandemir, Anand Sivasubramaniam, and Chita Das. Holoar: On-the-fly optimization of 3d holographic processing for augmented reality. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 494–506, 2021.