The Three Flatland Problem

HUD content for Augmented Reality Multiplicative Light Field Displays

George Hellouin de Ménibus

Delft University of Technology

TU Delft

The Three Flatland Problem

HUD content for Augmented Reality Multiplicative Light Field Displays

by

George Hellouin de Ménibus

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday September 18, 2025 at 10:00 AM.

Student number:	5048354	
Project duration:	March 1, 2025 – September, 2025	
Thesis committee:	Prof. dr. ir. R. Marroquim,	TU Delft, supervisor
	Prof. Dr. P. Kellnhofer	TU Delft
	Prof. Dr. M. Zuniga	TU Delft

Special thanks to: Dr. Rafael Romeir, Sarah Quinones (maintainer of faer)

Cover:	Through the Looking Glass by Alexandra Bamberger
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld and George Hellouin de Menibus

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Contents

1

Introduction

Augmented reality (AR) is a wide area of ongoing research. One of AR's promising developments is the delivery of critical information at just the right time. Head-Up displays, originally referring to systems that can deliver information to pilots without them needing to move their head, are a natural fit for AR. From logistics [17] to medicine [9], AR has shown the capacity to enhance human operators by delivering important information exactly when it is needed. For example, previous research in air traffic control showed a great increase in situational awareness, especially when dealing with low visibility conditions [35].

Surveying 3D technologies and approaches, light field displays (LFDs) stand out as a potential foundation for AR. LFDs present images with depth by accounting for more than one viewpoint. Previous work has leveraged this to achieve better depth perception for VR applications [21]. Using light field displays for AR is a nascent field, with many unique solutions being researched [41]. LFDs offer near-eye form factor similar to smart glasses [41], beating out traditional head-mounted displays (HMD). Additionally, LFDs offer greater depth perception than smart glasses. Most HMDs and smart glasses decouple focus cues and vergence, which can lead to discomfort and eye fatigue.

Several AR applications use simple HUDs, primarily projecting 2D elements (e.i. outlines, text, glyphs) to deliver critical information [17] [9]. Despite this, most approaches still employ complex 3D rendering techniques to display the content. Our approach, by contrast, leverages the content's 2D nature to achieve lower sampling times, more efficient memory representation and can be extended to support 2D animated content. We build on top of work in light field displays, allowing us to maintain correct foces cues and stereoscopy.

This work is possible thanks to advances in see through displays. Older research created the illusion of see-through by feeding the output of a camera to an LCD screen [7] and modifying the image for the AR enhancement. The latency of images created discomfort, as the movement of the hands does not match what is seen. Previous approaches in AR light fields experimented with see through displays, but the field lacked maturity, leading to poor results [18]. Ten years later, transparent displays have come a long way. New OLED displays offer high transparency [31], and have been used to display 2D content as smart windows or dashboard [30].

In our configuration, there are two stacked panels and one target content texture, which we collectively refer to as the three *flatlands*. We introduce a new methodology for decomposing a 2D UI element across our panels for augmented reality applications, particularly HMDs. We build on approaches used in LFDs to break down content between panels, as well as their findings for a minimal number of panels. Following that, we test this approach using a simulation, leaving the work of building a device that uses this approach as future work. Next, we compare our approach with Huang et al. [21] stereoscopic approach, as it resembles ours most closely.

The Road Not Travelled

This work originally aimed to use smart windows to create 3D effects from 2D content, instead of HMDs. However, this proved to be challenging, and initial results were discouraging. The approach developed there can still be found in Section 4.2 and was the foundation for the solution we arrived at.

The Shape of the Problem

Our goal is to efficiently display critical information through a HUD. This element is 2D but should look like it's at a certain distance, as if it were occupying physical space.

Examples of these HUD-like elements are quite common in video games:

(a) Screenshot A

(b) Screenshot B.

(c) Screenshot C

(d) Screenshot D.

Figure 1.1: Four screenshots taken in the game Abiotic Factor. Some HUD elements are fixed on screen, lower left, while others, highlighted by a red circle, are free-floating. They are perceived as occupying 3D space as they move when the observer moves.

The Shape of a Solution

In this work, we focus on the theoretical and software implementation of how to achieve this 3D effect. We do not venture into hardware-related questions. However, we assume a similar hardware implementation as *The Light Field Stereoscope* [21]. We assume the panels to have a high level of transparency. A perfect solution would allow us to make HUD elements have the correct depth, such that users can place the texture at a position in the real world.

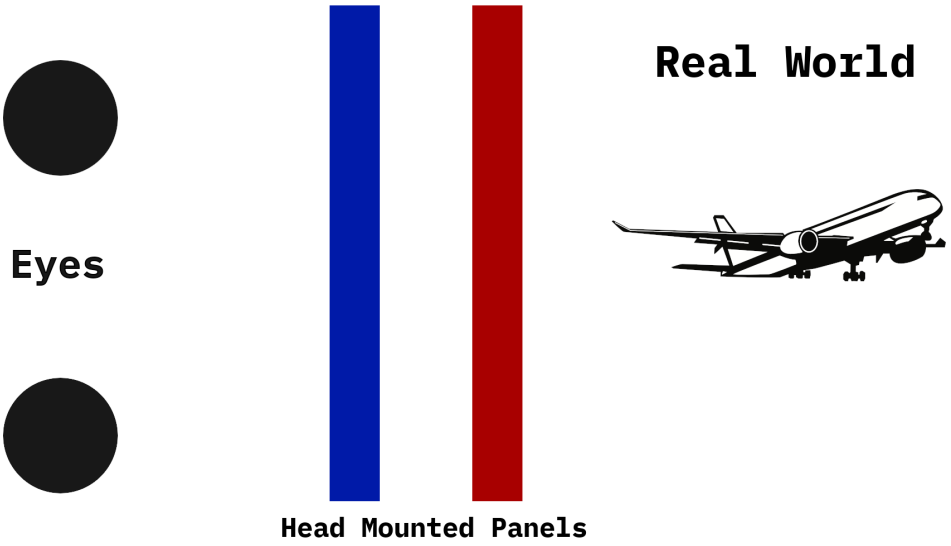


Figure 1.2: Head-Mounted Display outlook

Figure 1.3: This airplane is outlined and labelled as it lands. A control tower operator would have no problem identifying the craft at a glance. Modified from Wikipedia[32]

2

Background

This chapter details the general theory upon which our research depends. We examine the principles behind 3D displays and how to capitalize on those principles. We then discuss light, light fields and approaches to displaying them. Lastly, we finish the chapter by discussing advances in display technology that we presume will keep improving.

2.1 3D displays

Successful 3D displays seek to induce stereopsis: the sensation that objects have depth and objects have different distances from each other [12]. The human vision system relies on two mechanisms for stereopsis: binocular depth vision (the difference between both eyes) and monocular motion vision (the difference when eyes move around).

Binocular disparity can be achieved by presenting each eye with a different image. This is why most 3D displays rely on some technique to discriminate between eyes.

Motion vision is composed of two parts: optical flow and parallax. Optical flow comes from the patterns produced by the relative motion between an observer and a scene. Addressing optical flow is outside the scope of this work. Parallax stems from the difference in apparent position of an object seen from different positions. The pupil of the eye can move around, letting the eye observe an object at different positions with no head movement. Animals lacking round eyes, such as pigeons, rely on head movement instead of eye movement to derive motion parallax [40].

If we want to capture parallax, we need to consider our eyes as more than a pinhole camera. Instead, we will model the eye as an eyebox: a cloud of positions where the pupil can be. This allows us to replicate motion parallax from eyeball movement, as well as correct blurring when the eyes are looking elsewhere. An illustration of an eye box can be seen in Figure 2.1.

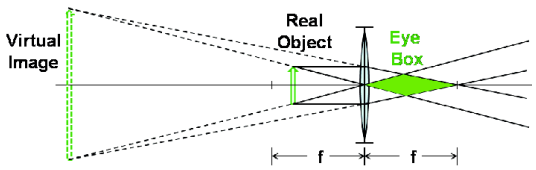


Figure 2.1: Eyebox for near eye display. Taken from Varshneya et al.[39].

We separate 3D approaches based on whether they use glasses/other eyewear or not. This distinction has important implications for what we can assume and how we can display our images.

Head-Mounted Displays

Head-Mounted Displays (HDM) are not new. The first efforts in 3D imaging can be attributed to Charles Wheatstone [1], who in 1838 investigated binocular vision and invented the first stereoscope. More than 100 years later, Ivan Sutherland and Bob

Sproul[33] would create the first HMD. The underlying idea was the same: present each eye a different image to create the illusion of a third dimension. Despite the same theoretical basis, modern VR headsets barely resemble Sutherland and Sproul's creation. The second wave of VR, starting in 2013, has introduced a wide range of consumer products that have found industrial applications [26]. This new generation is not without its limits. Long usage of headsets can cause eye fatigue [34], as well as motion sickness. Much work has been done to improve this, but the limitations still exist, notably the vergence-accommodation conflict [23].

Glasses-Free

Glasses-Free is an alternative paradigm for 3D imaging. As the name implies, it is defined by its lack of headwear and covers a wide range of techniques. Although not as old as head-mounted displays, Frederic E. Ives patented the first glasses-free stereoscope in 1903 [2].

The most successful approach is parallax barriers [22]. It's the technique employed by 3DS to achieve glasses-free 3D. By placing the barrier in front of an image source and controlling which image is projected to which eye, you can create stereoscopic views. However, parallax barriers require the observer to be at a predetermined location.

Another approach is holography/volumetric displays. These displays have numerous strengths compared to conventional flat screen approaches. By providing realistic focal depth, motion parallax, and vergence, they overcome the vergence-accommodation conflict that HMDs suffer from [13].

2.2 Light Physics

Light's properties dictate the modelling approach we can use. Depending on the context, light can be multiplicative or additive. When we are discussing transmittance, light perception is multiplicative. It takes a value between 0 and 1, with 0 being total opacity and 1 total transparency. Light perception from multiple sources on the surface is additive. Multiple projectors projecting on top of one another will be perceived as the sum of their results.

In this work, we focus on a multiplicative approach. A brief discussion relating to additive approaches can be found in section 3.2.

We will not concern ourselves with questions of polarity, as we assume all panels polarize light the same way.

2.3 Lambertian Surfaces

Lambertian surfaces reflect light equally in all directions [5]. They will not exhibit view-dependent effects, such as specular highlights. Very few materials exhibit perfect Lambertian properties in the real world, but it remains a relevant model in computer graphics, image processing, and computer vision. The guarantees from modeling HUD elements as Lambertian are crucial to our method.

2.4 Light Fields

Light fields are a theoretical construct that describes all light flowing in every direction through every point in space. All possible light rays can be described as the following five-dimensional function:

$$L(x, y, z, \theta, \phi),$$

where ϕ is the heading angle in the XY-plane, and θ is the angle between the light ray and the positive Z-axis. As visualized in Figure 2.2, we can see that all light rays are a point in space and two angles to give us a direction.

2.5 Light field displays

Light field displays refer to displays that seek to capture the light field of a particular scene. Several approaches exist, which are further developed in chapter 3. The most

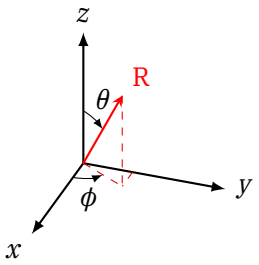


Figure 2.2: Visualization of light ray R defined by the function $L(x, y, z, \theta, \phi)$

straightforward approach is stacking LCD panels and relying on the position of the observers to see different combinations of pixels between the panels. This creates the illusion of 3D and depth, as observers at different positions will see different images because, depending on their position, the pixels on different panels will interact differently, as illustrated below.

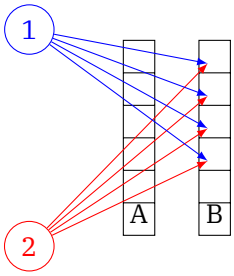


Figure 2.3: Two different observers looking through two transparent panels. Both observers see the entire back panel B, but not the entire front panel A.

Light field displays have been used for both HMDs and glasses-free approaches. They offer some unique advantages over their HMD counterparts, notably a wider range of focus cues. this is crucial at tackling the vergence-accommodation conflict, a source of major discomfort in headset applications.

The main challenge for light field displays is to decompose the light field across these multiple screens. When building a headset, it is possible to optimize the decomposition for each eye, as their positions in relation to the screen are known. Moreover, pixel combinations that are unlikely to come up might matter less for the computation. If the headset has eye tracking, it is possible to further optimize for where the retina is directly looking at. By contrast, a light field display that is trying to simulate a large multi-user display (e.g., an aquarium) might prioritize multiple viewing angles, even if it introduces noise.

Additionally, light field displays often suffer from lower perceived resolutions due to their decomposition of content.

2.6 Transparent Displays

Improvements in transparent displays are a primary motivator for our work. There have been considerable efforts [25] into developing transparent display technology, with recent developments going all the way to 20% transmittance [36]. By using ambient light as a backlight, the energy footprint of such screens is reduced, and a wide range of applications opens up. Notably, such improvements have been used in architecture in pursuit of low-emission buildings [45].

When leaving the world of LCDS, there are OLED displays that have shown up to 64% transmittance [31], enough for information displays on windshields and dashboards. Lastly, there have been developments in optical-grade silicon carbide lenses[42], promising even greater returns, but today they remain expensive to manufacture.

3

Related work

This chapter focuses on developments in light field displays (LFDs). We divide approaches based on their relationship with light and explore developments in multiplicative and additive LFDS. We identify multiplicative LFDs as having some important advantages for our problem. However, little work has been done in applying multiplicative approaches to 2D content. By contrast, additive approaches have been used for HUD content but are ill-suited for our goals of high autonomy and low power.

3.1 Multiplicative Light

To our knowledge, the first multiplicative approach to light field displays comes from Wetzstein et al. [15]. Building on top of work in volumetric displays, they developed an approach that used stacked attenuation layers to recreate a 4D light field. The content of the layers came from a tomographic[1] decomposition of the scene, while the coefficients of each layer are determined by minimizing the error for a discrete set of rays. This produces high-resolution glasses-free 3D images with limited degrees of freedom and is restricted to a static image. Douglas et al [14] take this approach further by replacing the attenuation layers with polarized light fields, reducing artifacts and increasing resolution and depth of field, but similar limitations remain.

These downsides were addressed with tensor displays [16]. Tensor displays use a stack of time-multiplexed, light-attenuating layers illuminated by a directional backlight. This improves depth of field and FOV significantly, all while remaining glasses-free.

So far, all this work was done with glasses-free in mind. However, Huang et al [21] demonstrate that not only could this be used for VR, but that in easing the requirements, they arrived at a simpler solution. Their approach uses only two stacked panels and no time-multiplexing. They use light fields to address one of VR's great challenges: the vergence-accommodation conflict. Most VR headsets force users to maintain a fixed focal distance while varying the vergence angle of their eyes. Dubbed *The Light Field Stereoscope*, this device presents to the observer a correct retinal blur, allowing the user to freely focus their eye within the scene.

To achieve correct focus cues, they modelled the eye as an eye box, sampling multiple positions around the eye to accommodate multiple focus cues. Determining the optimal eyebox for near eye displays is an area of ongoing research [39].

The authors of *The Light Field Stereoscope* lament the blur imposed by see-through LCDs on physical objects, pointing to that as a major roadblock to augmented reality implementations. Such limitations can be seen in Miamo and Fuchs' [18] proposed solution. It is difficult to make out much of the real world due to the blurring. Additionally, Miamo and Fuchs' work is primarily concerned with the hardware implementations and not with optimizing around the specifics of AR content.

However, since the publication of *The Light Field Stereoscope* a decade ago, there have been many advances in see-through head-mounted displays [41]. When looking for

[1] Tomography is an imaging by sections/slices often used in medical imaging.

literature on HUD elements for AR LFDs building on top of these advances, we find additive approaches but no multiplicative ones.

3.2 Additive Light

There are approaches within additive light fields that are similar to the multiplicative layers. Narian et al. [24] suggested a multifocal display that combines layers additively thanks to mirrors, prisms, and beamsplitters. Compared to Wetzstein et al. [15], the goal of this approach is to support accommodation and defocus, drawing primarily from image processing techniques. However, it requires a complex setup with a bulky form factor. The multi-plane display is larger than most PC monitors.

Lee et al. [27] took a similar concept and applied it to augmented reality with no headset. They combined the additive fields with holographic optical elements (HOE). Compared to previous approaches, HOE layers have little diffraction, increasing image quality and configuration. This approach produces interesting images but suffers from a similar form factor as the previous one. Additionally, the projected scenes are made from slices of the 3D objects. This approach would not work for 2D content without heavy modification.

Jan et al. [28] build on this additive approach by placing it on a head-mounted display. Compared to AR systems of the time, it provides high contrast, resolution, and focus cues in a large depth range. However, the use of HOE makes this approach ill-suited for 2D content, as generating holographic elements from 2D content is an area of ongoing research.

Lee et al. [37] expand the additive light field approach to HUP content, specifically for automotive drivers. Their work discusses at great length how to build and implement such a system. Given that their scope is limited to automotive drives and does not rely on head-mounted technology, their approach is of limited use to us.

While on the whole, additive approaches have been applied to AR more than multiplicative, none seek to address the original challenge we laid out in the introduction: the poor autonomy from power consumption. None leverage the low-power profile of modern see-through displays, as their image generation/display is built on top of projectors, not displays. By contrast, multiplicative methods seem more applicable to modern see-through displays.

4

The Three Flatlands: Method

In this chapter, we propose our method for decomposing two-dimensional content across the two transparent panels of our head-mounted display. We model the eyes as an eyebox: a cloud of potential observing positions in a sphere in space. This means we should develop a method to decompose our target texture across the panels that can take arbitrary observers. Once we have captured the general case, we can move forward to our specific eye box case.

We first define *The Three Flatlands*, as well as our core assumptions. Subsequently, we begin with a brute force method that will reveal some properties of the problem space. Next, we introduce our current method, first in two dimensions and then in three. Following that, we address a unique challenge: the void result. Previous methods have not had to consider how to handle uninitialized pixels. Lastly, we discuss how to bring this back to our head-mounted display.

4.1 Core Assumptions

An observer will perceive light based on the light rays in a scene. The values of these rays depend on material properties, such as refraction, transparency, and reflection. Our input is a target texture T that we want an observer looking out of our panels to see. This target texture is virtual, but the panels are physical. The texture is at an arbitrary location, and the observers are known ahead of time. Because our target is a 2D texture, it is a Lambertian surface. It reflects light independently of the incident direction. We can control the individual pixel transparency on each panel. A visualization of the setup can be seen in Figure 4.1.

We deal exclusively with pixel opacity, a number between 0 and 1, and will work in grayscale. Content is transformed to grayscale, with white being transparent and black being fully opaque. This is necessary for the texture color values to match the physics.

We need to account for two eye boxes that are known ahead of time. Therefore, we need a method that can handle multiple viewpoints: two eyeboxes, each with multiple sample points inside of them. For modeling, we do not differentiate between eyeboxes, we need to find a solution that works for an arbitrary number of viewpoints. This also allows changing the granularity of the eyeboxes.

4.2 Brute Force

Let's start by solving the problem theoretically. As developed in section 2.5, light fields are a five-dimensional function that describes all light rays in a scene. Given our use of stacked panels, we can reduce this space to four dimensions. All rays that our panels can represent must intersect with both of them. It follows that each ray can be described as two points, one on each panel. Given that a point on a plane can be described

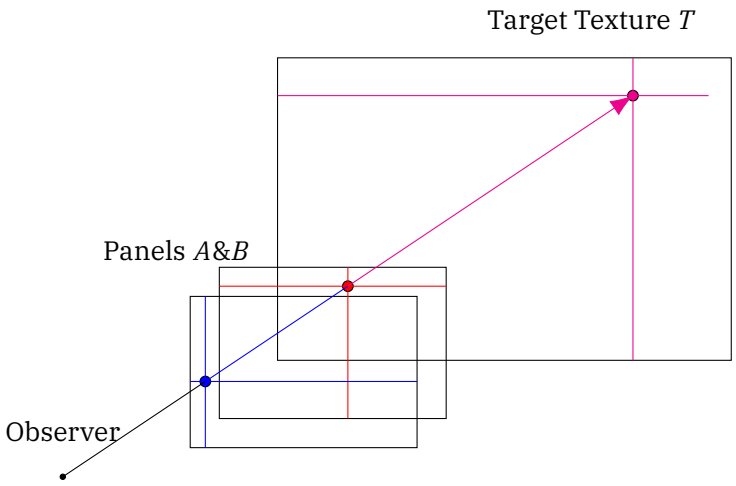


Figure 4.1: *The Three Flatlands* corresponding to the target texture, T, which is virtual, and the panels A and B, which are physical. Observer will see a combination of content on Panels A & B.

with two coordinates, the two points can be described with four coordinates. We illustrate this in Figure 4.2.

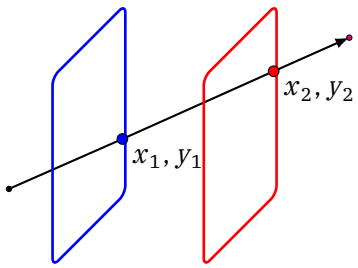


Figure 4.2: Two panels. Ray can be described by four coordinates, corresponding to 1 point in each panel

Our panels have limited resolution, which we can describe as a discrete space. This means there is a finite number of possible coordinate combinations that describe all rays that the panels can recreate. These are the pixels on each panel. We can only capture rays that go through the center of pixels in both panels. Therefor, the size of the pixels imposes a limit to the rays we can recreate. All rays that go through the same pixels will be binned into the same ray. This imposes a limitation on the light field we can recreate, as seen in Figure 4.3.

A naive solution would be to sample the entire light field space and use this to build the content on both our panels.

We can model all possible combinations of pixels on the panels as the matrix product of two vectors. Each vector is composed of all pixels in one of the panels. We can fill this output matrix by sampling our scene, that is, constructing a ray for every combination and recording its value in the corresponding entry in the matrix. Then, we can use Non-negative matrix factorization (NMF) to reconstruct the content. Figure 4.4 shows how we can apply NMF to our problem.

NMF ensures our content is not negative (no negative light intensity), but values across iterations need to be capped, as we are limited to the 0 to 1 range. NMF has been used as a way of solving for similar light field problems [11]. We modify Lee and Seung's multiplicative update rule [8] to cap values at 1. We get the following update rules:

$$\mathbf{H}^{n+1}_{[i,j]} \leftarrow \max(1, \mathbf{H}^{n}_{[i,j]} \frac{((\mathbf{W}^n)^T \mathbf{V})_{[i,j]}}{((\mathbf{W}^n)^T \mathbf{W}^n \mathbf{H}^n)_{[i,j]}})$$

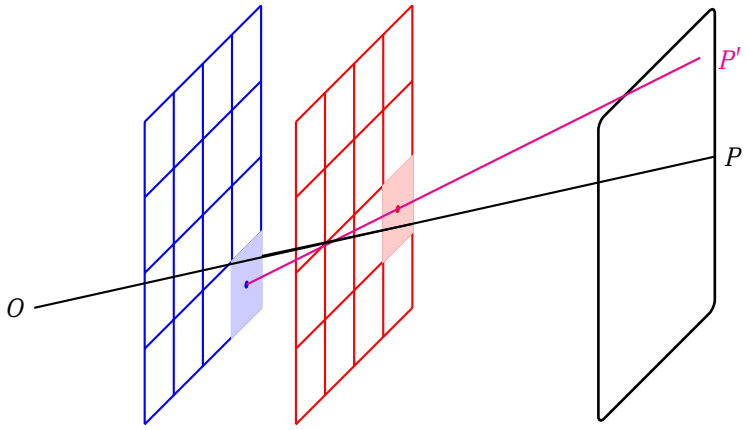
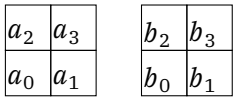


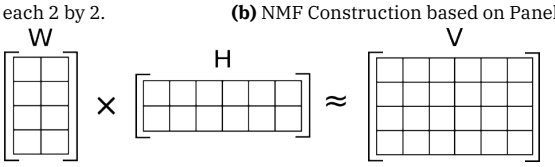
Figure 4.3: Limitations imposed by the panels' resolution. The ray from O to P will be binned into the ray ending at P'. Even with perfect panels, P' and P are not the same location and can be different colours, imposing a hard limit on what we can recreate.



(a) Two panels, A and B, each 2 by 2.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_0 * b_0 & a_0 * b_1 & a_0 * b_2 & a_0 * b_3 \\ a_1 * b_0 & a_1 * b_1 & a_1 * b_2 & a_1 * b_3 \\ a_2 * b_0 & a_2 * b_1 & a_2 * b_2 & a_2 * b_3 \\ a_3 * b_0 & a_3 * b_1 & a_3 * b_2 & a_3 * b_3 \end{bmatrix}$$

(b) NMF Construction based on Panels A and B



(c) NMF Illustration taken Wikipedia [19]. We approximate the values of W and H based on V

Figure 4.4: NMF Example

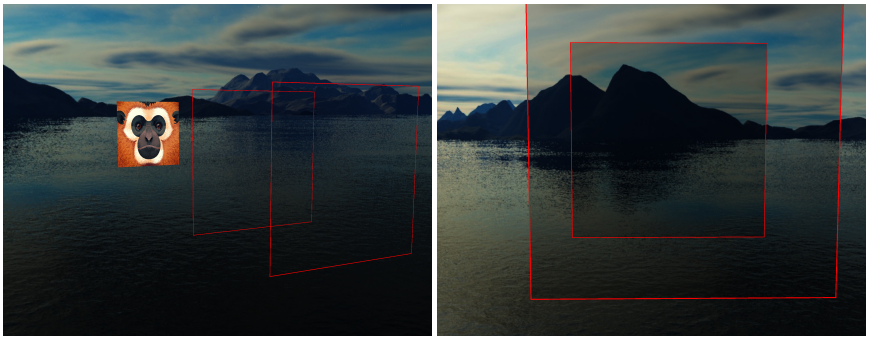
$$\mathbf{W}^{n+1}_{[i,j]} \leftarrow \max(1, \mathbf{W}^{n}_{[i,j]} \frac{(\mathbf{V}(\mathbf{H}^{n+1})^T)_{[i,j]}}{(\mathbf{W}^{n}\mathbf{H}^{n+1}(\mathbf{H}^{n+1})^T)_{[i,j]}})$$

This approach has many shortcomings, both practical and theoretical. The practical is that the matrix $\mathbf{V}$ grows at the power of $O(n^4)$. For example, given two square panels, each with 1000×1000 pixels, leads to a matrix with 1000^4 entries. Assuming 32 bits per entry, $\mathbf{V}$ requires 4 TB of memory.

The theoretical shortcoming is that we are sampling the entire light field space, with equal weight to all rays. Without resorting to multiplexing [22], we are limited in our possible representations. Additionally, rays that miss our target are assigned the value of 1.

Previous work has shown that errors in scene reconstruction can be optimized around [47]. While the overall error may remain the same, a clever method can optimize for the perceived image. However, this brute force method spreads the error out in such a way that we perceive a homogeneous gray. The solver gravitates towards slightly tinted screens where none of the target texture can be seen. A rendered example of the output of NMF can be seen at Figure 4.5

This brute force approach considers the entire light field reconstruction. However, our use case knows the positions of the observers, and therefore can restrict our ray consideration to only those that are observed. This means we can disregard pixel combinations that are not seen. Let's turn our attention to solving for arbitrary ray selected by our observers.



(a) Target texture and both panels **(b)** Content Produced by NMF

Figure 4.5: The output produced by NMF. The cubemap of the mountain acts as the physical world and is not sampled by NMF. Notice the disappearance of the target texture, as it gets lost to the noise of all rays that will not hit it.

4.3 Arbitrary Rays Definition

Let R denote the set of n_r light rays, with each ray being denoted as r_i, $i \leq n_r$. We have two panels, A, B and our target texture T. Each ray can intersect each element, A, B or T, exactly once, as there are no reflections or refractions. Since we assume our target texture to be a Lambertian surface, all rays that intersect T at the same point t_p report the same value. If a light ray r_p intersects A at pixel a_p, B at pixel b_p and T at t_p, this imposes the following constraint:

$$a_p b_p = t_p \tag{4.1}$$

Note that we multiply the values a_p and b_p because light is multiplicative.

The formulation of Equation 4.1 is restrictive. It needs to be computed for every different ray. However, we can modify it to profit from some invariance across different rays. We begin by replacing a_p, b_p, t_p with a vectorized representation of their pixels: $\vec{a}, \vec{b}$ and $\vec{t}$ to capture all different locations a potential ray intersects. Now, we can introduce a mapping from entries in $\vec{a}, \vec{b}$ to elements in $\vec{t}$ that no longer depend on p. For a single light ray, we build a vector for each element, $\vec{m}$, which encodes the pixel that the ray hits.

$$\vec{m} = \begin{pmatrix} 0 \\ \dots \\ 1 \\ \dots \end{pmatrix} \tag{4.2}$$

We now modify Equation 4.1 to account for all pixels in the scene:

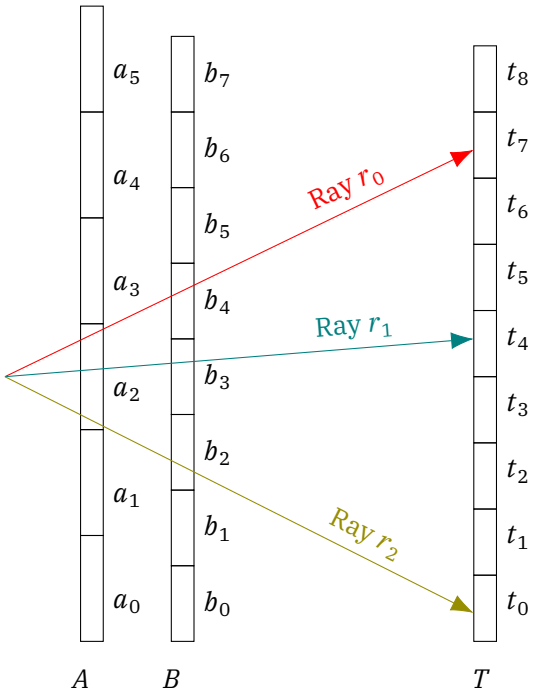
$$\vec{t} \cdot \vec{m}_t = (\vec{a} \cdot \vec{m}_a)(\vec{b} \cdot \vec{m}_b) \tag{4.3}$$

It follows that when considering different rays, we only need to modify the m vectors. While this new formulation is more flexible than Equation 4.1, it is still limited to one ray at a time. The vectors need to be modified for every ray. To capture the constraints of multiple rays, we need to extend Equation 4.3 to include each ray instead of modifying $\vec{m}$. We begin by considering the problem in two dimensions before moving on to three.

4.4 Two Dimensions

In 2D, A, B, T are pixel rows, with n_a, n_b, n_t number of pixels. Just as before, we can describe the content of A, B, T as the vectors $\vec{a}, \vec{b}, \vec{t}$. We want to introduce a way to map entries in each vector depending on the light rays, such that only relevant content is used to build constraints similar to Equation 4.3. We propose a matrix for each scene element that maps the relationship between rays and pixels. When these matrices are multiplied by their respective vectors, each row in the resulting vector will correspond to a constraint created by that ray.

We build the matrices as follows: Let $E \in (A, B, T)$. If Ray r_i intersects E at pixel e_b, then $\mathbf{M}_{e[i,b]}$ is 1. Otherwise, it's 0. This new Matrix, $\mathbf{M_e}$ can be thought of as combining multiple $\vec{m}_e$. It will have as many rows as there are rays, with each row corresponding to one ray's $\vec{m}_e$. Applying this to all our elements, we build matrices $\mathbf{M_a}, \mathbf{M_b}, \mathbf{M_t}$, where $\mathbf{M_a}$ has dimensions $n_r \times n_a$, $\mathbf{M_b}$ dimensions $n_r \times n_b$ and $\mathbf{M_t}$ dimensions $n_r \times n_t$. These mapping matrices allow us to transition from content space to the ray space. An example of this construction is shown in Figure 4.6



(a) View of three rays originating at one observer

$$\mathbf{M_a} = \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{M_b} = \begin{bmatrix} 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{M_t} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) The matrices that map the incidence of the rays

Figure 4.6: Construction of Mapping Matrices based on three rays

We can capture the previous constraint Equation 4.1 for all rays, as the matrices ensure we only multiply entries that have a ray relation between them:

$$\mathbf{M_t}\vec{t} \approx \mathbf{M_a}\vec{a} \circ \mathbf{M_b}\vec{b} \tag{4.4}$$

where $\circ$ denotes the Hadmard (element-wise) product.

This formulation leads to the following optimization problem:

$$\min_{\vec{a},\vec{b}} ||\mathbf{M_t}\vec{t} - \mathbf{M_a}\vec{a} \circ \mathbf{M_b}\vec{b}||^2, \tag{4.5}$$

Where all elements of $\vec{a}, \vec{b}$ are between 0 and 1.

In order to reason about the content of $\vec{a}, \vec{b}$, we will need to transform from ray space to content space. However, we have no guarantees that $\mathbf{M_a}$ or $\mathbf{M_b}$ are square matrices. Even if they were square matrices, inverting them risks introducing negative numbers.

We turn once more to NMF to solve this. The objective function stated in Equation 4.5 is not convex. By fixing one of the two matrices, we turn the function convex and can find a minimum using the gradient. We extend Lee and Seung's [8] update rules to include the mapping matrices into the solver. Applying the NMF methods gives us the following update rules:

$$\vec{a} \leftarrow \vec{a} \circ \frac{\mathbf{M_a^T}(\mathbf{M_t}\vec{t} \circ \mathbf{M_b}\vec{b})}{\mathbf{M_a^T}(\mathbf{M_a}\vec{a} \circ \mathbf{M_b}\vec{b} \circ \mathbf{M_b}\vec{b})} \tag{4.6}$$

$$\vec{b} \leftarrow \vec{b} \circ \frac{\mathbf{M_b^T}(\mathbf{M_t}\vec{t} \circ \mathbf{M_a}\vec{a})}{\mathbf{M_b^T}(\mathbf{M_b}\vec{b} \circ \mathbf{M_a}\vec{a} \circ \mathbf{M_a}\vec{a})} \tag{4.7}$$

In a loose sense, the transpose plays a similar role in these rules as the pseudoinverse does in unconstrained least squares. It allows us to map from the data space (rays) back to the coefficient space (content). Unlike the pseudoinverse, it is not exact.

4.4.1 Update rule Simplification

The update relies on treating one of the two values as a constant, allowing us to employ a gradient descent technique. This lets us update our estimate for $\vec{a}$ and $\vec{b}$ in such a way that the error in each iteration never grows. For the sake of simplifying this elaboration, let us replace the mapping matrices with the identity matrix and assume all vectors have the same size. This means that the new optimization function is:

$$\min_{\vec{a},\vec{b}} ||\vec{t} - \vec{a} \circ \vec{b}||^2$$

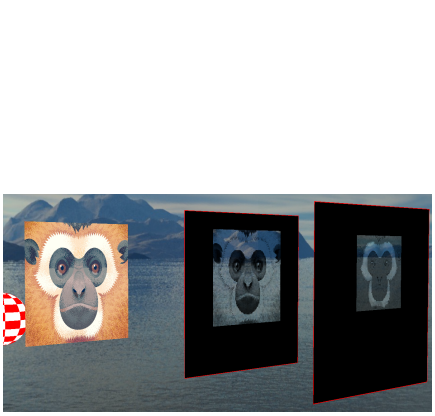
Likewise, we simplify our update rule for $\vec{a}$:

$$\vec{a} \leftarrow \vec{a} \circ \frac{(\vec{t} \circ \vec{b})}{(\vec{a} \circ \vec{b} \circ \vec{b})} \tag{4.8}$$

In this simplification, the effects of the update rule are clear. If an entry a_i in $\vec{a}$ is too large, $a_i \times b_i > t_i$, it will decrease proportionally. If an entry is too small, it will increase proportionally as well. An exact entry will result in no change. For a longer discussion on NMF, we recommend Blondel et al [10], which delves into many of the proofs for non-negative matrix factorization.

4.4.2 Generated Images

We can extend this approach to 3D by treating our content as if it were 2D. We can vectorize our panels and target texture. An example resulting image can be seen in Figure 4.7 Compared to the brute force approach, the produced image is much closer to our target. Unsampled pixels default to 0, which is black. This is a problem we will return to later. However, we have not escaped the memory challenges of the brute force method.



(a) The target texture, the gibbon, and the two panels

(b) The reconstruction seen through both panels

Figure 4.7

In this case, $\vec{a}$ would be a vector of size $w_a \times h_a$, where w_a, h_a are the width and height of A. The matrix $\mathbf{M_a}$ would have dimensions $n_r \times (w_a h_a)$. Given that for one viewpoint, we require at least as many rays as there are pixels in our target texture, the matrix $\mathbf{M_a}$ would scale as $w_t h_t \times w_a h_a$. For a panel with 1k pixels by 1k pixels, and a target texture of equal resolution, this would mean $1,000^4$ entries. Assuming 32-bit floating-point representation, that would mean storing $\mathbf{M_a}$ would take 4000 GB, or 4 TB, the same size as our brute force strategy with V. This is because while we have found a way to optimize for our desired rays, we are representing a one-to-one relation between rays

and pixels as a massive matrix that must consider all possible combinations. Unlike V, $\mathbf{M_a}$, is sparse, and can be stored as such.

Nonetheless, it feels inefficient to represent so much of the light field, when we are only interested in a small part.

4.5 Three dimensions

To tackle the memory challenge, we need to move to a different representation of the problem. For this section, we will begin by considering a single viewpoint before moving on to multiple viewpoints.

Going back to our light field theory, we observe that if we have a fixed position, (x, y, z), all rays originating or passing through it can be described with two dimensions: θ, ϕ. These two dimensions are independent. As the brute force section laid out, we are working in a discrete space. We can discretize θ and ϕ by placing a gridded plane at an arbitrary distance and binning rays together that intersect the same row or column of our plane. This is similar to image plane discretization in ray tracing:

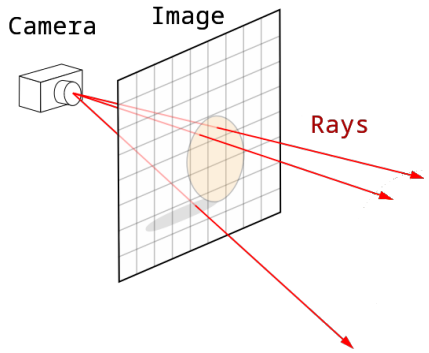


Figure 4.8: Discretizing θ and ϕ. Similar to raytracing, we capture the different angles with a gride placed at some distance. Image modified from Wikipedia.[1]

Let us call our discretized spaces X and Y such that each ray corresponds to an entry x_r and y_r. Rays that share coordinates share geometric properties. They will have the same angle for one of the two dimensions.

Instead of thinking of rays as selecting pixels, let's think of their coordinates, x_r and y_r, as selecting rows and columns. Given that both our panels and target image are 2D discreet planes, whose pixels can be addressed as a specific row and column, we can reformulate our problem to use x_r and y_r.

We begin by reexpressing our original constraint, Equation 4.3, using 2D representations of our data. We construct A, B, T as matrices $\mathbf{A}, \mathbf{B}, \mathbf{T}$, with dimensions $h_a \times w_a, h_b \times w_b, h_t \times w_t$.

Just as we did in Equation 4.2 if we only have 1 ray, we could represent this as two vectors, $\vec{x}$ and $\vec{y}$ that select a row and a column from either our panels or target. This means we can rewrite our original constraint (Equation 4.1) as:

$$\vec{y}_t \mathbf{T} [\vec{x}_t]^T = (\vec{y}_a \mathbf{A} [\vec{x}_a]^T)(\vec{y}_b \mathbf{B} [\vec{x}_b]^T) \tag{4.9}$$

Just as before, we can capture this for multiple rays by joining all these mapping vectors. Let $E \in (A, B, T)$. If ray (r, w) intersects E at pixel e_v, which sits at row k and column l , then $\mathbf{M_{e,y}}[w, k]$ is 1 and $\mathbf{M_{e,x}}[r, l]$ is also 1. Otherwise, it's 0. In a way, we are decomposing $\mathbf{M_e}$ into two matrices, a vertical and horizontal component.

Applying this to each of our flatlands, we get much smaller matrices. For instance, $\mathbf{M_{a,y}}$ with dimensions $y_r \times h_a$ and $\mathbf{M_{a,x}}$ with dimensions $x_r \times w_a$.

Going back to our earlier example of 1k panels, $\mathbf{M_{a,y}}$ and $\mathbf{M_{a,x}}$ would have size 1000^2. Assuming the same 32-bit representation, this would take 8 MB instead of 4 TB.

An illustrative example of this construction can be seen in Figure 4.9.

[1] https://en.wikipedia.org/wiki/Ray_tracing_(graphics)#

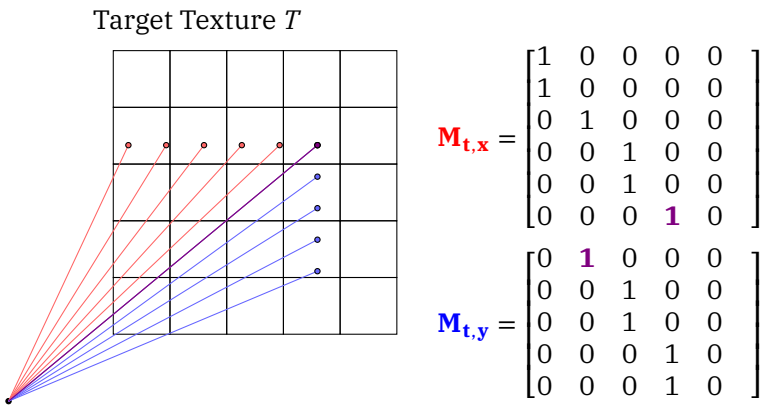
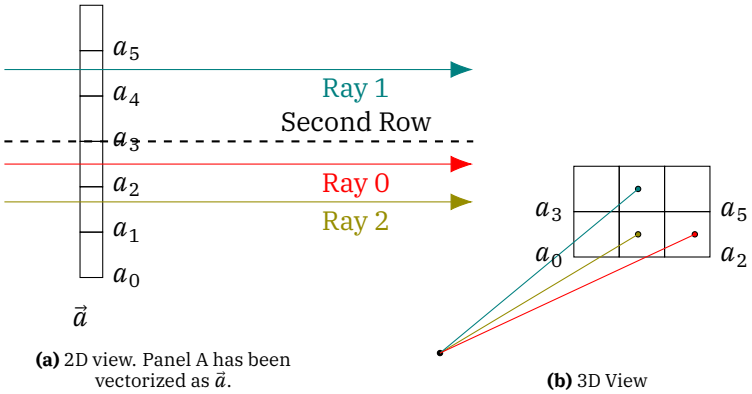


Figure 4.9: Construction of matrices based on rays. Notice how the same ray appears in both matrices

This construction has some differences compared to the previous approach. Figure 4.10 explores the differences between the previous 2D approach and this new 3D approach by building $\mathbf{M_{a,x}}$, $\mathbf{M_{a,y}}$ and, $\mathbf{M_a}$ for the same scene. As can be seen in the figure, rays can be redundant, a topic we will explore in section 4.6, once we have extended this approach to multiple observers.



(a) 2D view. Panel A has been vectorized as $\vec{a}$.

(b) 3D View

$$\mathbf{M_a} = \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{M_a}\vec{a} = \begin{pmatrix} a_2 \\ a_1 \\ a_4 \end{pmatrix}$$

$$\mathbf{M_{a,y}} = \begin{bmatrix} 0 & \mathbf{1} \\ \mathbf{1} & 0 \end{bmatrix}, \mathbf{M_{a,x}} = \begin{bmatrix} 0 & 0 & \mathbf{1} \\ 0 & \mathbf{1} & 0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \end{bmatrix}$$

$$\mathbf{M_{a,y}} \mathbf{A} \mathbf{M_{a,x}^T} = \begin{bmatrix} a_4 & a_5 \\ a_1 & a_2 \end{bmatrix}$$

Figure 4.10: The same scene being represented by both methods. For the 3D approach, ray 2 is redundant, as it is already captured by rays 1 and 0, shown as brown in the matrices. Likewise, a_5 is captured despite no ray sampling it.

With this structure, the new approximation becomes:

$$\mathbf{M_{t,y}} \mathbf{T} \mathbf{M_{t,x}^T} \approx \mathbf{M_{a,y}} \mathbf{A} \mathbf{M_{a,x}^T} \circ \mathbf{M_{b,y}} \mathbf{B} \mathbf{M_{b,x}^T} \tag{4.10}$$

The optimization problem now is:

$$\min_{\mathbf{A},\mathbf{B}} ||\mathbf{M_{t,y}} \mathbf{T} \mathbf{M_{t,x}^T} - \mathbf{M_{a,y}} \mathbf{A} \mathbf{M_{a,x}^T} \circ \mathbf{M_{b,y}} \mathbf{B} \mathbf{M_{b,x}^T}||^2 \tag{4.11}$$

Which can be solved with the alternating update rules:

$$\mathbf{A} \leftarrow \mathbf{A} \circ \frac{\mathbf{M_{a,y}^T}(\mathbf{M_{t,y}} \mathbf{T} \mathbf{M_{t,x}^T} \circ \mathbf{M_{b,y}} \mathbf{B} \mathbf{M_{b,x}^T}) \mathbf{M_{a,x}}}{\mathbf{M_{a,y}^T}(\mathbf{M_{a,y}} \mathbf{A} \mathbf{M_{a,x}^T} \circ \mathbf{M_{b,y}} \mathbf{B} \mathbf{M_{b,x}^T} \circ \mathbf{M_{b,y}} \mathbf{B} \mathbf{M_{b,x}^T}) \mathbf{M_{a,x}}} \tag{4.12}$$

$$\mathbf{B} \leftarrow \mathbf{B} \circ \frac{\mathbf{M}^{\mathbf{T}}_{\mathbf{b},\mathbf{y}}(\mathbf{M}_{\mathbf{t},\mathbf{y}}\mathbf{T}\mathbf{M}^{\mathbf{T}}_{\mathbf{t},\mathbf{x}} \circ \mathbf{M}_{\mathbf{a},\mathbf{y}}\mathbf{A}\mathbf{M}^{\mathbf{T}}_{\mathbf{a},\mathbf{x}})\mathbf{M}_{\mathbf{b},\mathbf{x}}}{\mathbf{M}^{\mathbf{T}}_{\mathbf{b},\mathbf{y}}(\mathbf{M}_{\mathbf{b},\mathbf{y}}\mathbf{B}\mathbf{M}^{\mathbf{T}}_{\mathbf{b},\mathbf{x}} \circ \mathbf{M}_{\mathbf{a},\mathbf{y}}\mathbf{A}\mathbf{M}^{\mathbf{T}}_{\mathbf{a},\mathbf{x}} \circ \mathbf{M}_{\mathbf{a},\mathbf{y}}\mathbf{A}\mathbf{M}^{\mathbf{T}}_{\mathbf{a},\mathbf{x}})\mathbf{M}_{\mathbf{b},\mathbf{x}}} \tag{4.13}$$

4.5.1 Multiple Viewpoints

Until now, we have ignored the question of dealing with multiple viewpoints. This introduces some additional challenges into our approach.

When we introduce multiple observers, we need to introduce a way to address rays while maintaining our ray separability. Two rays starting at the same location on the target, headed towards different observers, need to have different coordinates. These two rays do not necessarily share the same trajectory and may intersect different rows or columns of the panels. If they have the same x and y, we would treat them as the same ray and violate one of our core assumptions: that a ray can only intersect with a panel or texture once, at one discrete location.

The simplest approach would be to extend the width and height by the number of observers. Geometrically, this is inelegant, as it breaks the simple discretization of rays. However, it can work. This results in a ray space that grows exponentially. Each viewpoint will increase the dimensions of the ray space by its ray width and ray height. Assuming all viewpoints cast $h \times w$ rays, the final ray space will have size: $n^2 \times h \times w$.

It is possible to produce ray indexes by keeping the observers in a list. Assuming each observer casts x_t, y_t rays, and x, y are the original coordinates of the ray if there was one view point, then the new coordinate can be calculated according to Equation 4.14 and Equation 4.15.

$$x_i = x + x_t * o; \tag{4.14}$$

$$y_i = y + y_t * o; \tag{4.15}$$

However, this approach to ray indexing creates combinations of rows and columns that will never be active. We have sacrificed size to preserve the geometric relationship between rays that share coordinates. Figure 4.11 shows the rapid speed at which this space grows.

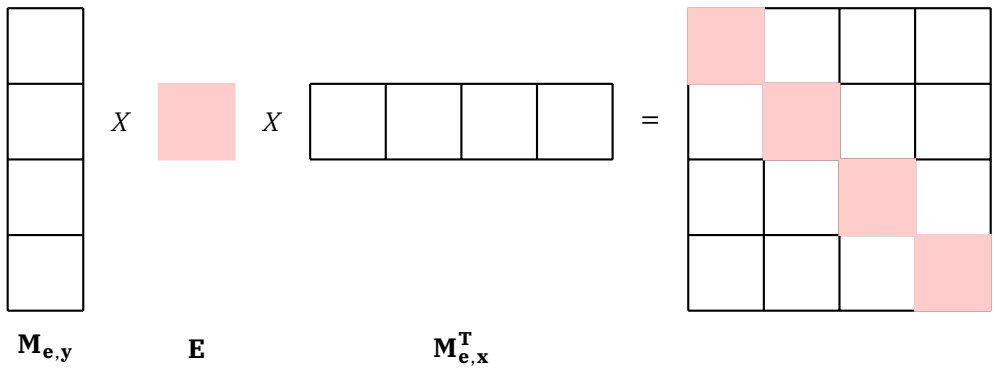


Figure 4.11: A representation of the ray space created by multiplying the mapping matrices with the content. Each grid corresponds to the mapping matrix of an observer. With only 4 view points, the ray space is 16 times greater. However, when transforming back to content space, only the red diagonal will be considered. For the sake of illustrating this problem, we assume there are as many rays as pixels in the content, producing a ray space of identical size.

Instead, let us treat each observer as a dimension, such that each ray,3 coordinates: x, y, and o for the observer. Our update rule relies on transforming content space to ray space and then back to content space through the mapping matrices. By building independent mapping matrices for each viewpoint, we can combine all matrices in content space and avoid useless computations in ray space.

This means every viewpoint has two mapping matrices for each element. Transforming to the ray space of an individual observer would be written as: $\mathbf{M}_{\mathbf{e},\mathbf{y},\mathbf{o}}\mathbf{E}\mathbf{M}^{\mathbf{T}}_{\mathbf{e},\mathbf{x},\mathbf{o}}$ To capture the entire ray space, we need to introduce tensors. We use tensors to capture all mapping matrices as well as the entire ray space. This means each element has two mapping tensors: X and Y.

Using Einstein summation notation, we can express the transformation from content space to ray space with the following operation:

$$R_{n,h,w} = Y_{n,h,k} C_{k,m} X_{n,m,w} \tag{4.16}$$

R is the tensor representing ray space, Y is the mapping tensor for columns, and X is the mapping tensor for rows. n is the number of viewpoints, h is the height of the rays, w is the width of the ray, k is the height of the content, and m is the width of the content. Note that X is transposed. Expressed as matrices, this can be understood as: $\forall o \in O, [\mathbf{M_{e,y,o}} \mathbf{EM^{T}_{e,x,o}}]$ where O is the set of observers and e is some element (panel or target).

We can express the transformation from ray-space to content-space with the following operation:

$$C_{k,m} = Y_{n,k,h} R_{n,h,w} X_{n,w,m} \tag{4.17}$$

Note the implicit summation over the number of viewpoints and the transpose of Y. Expressed as matrices, this is similar to: $\sum_{i=0}^{o} \mathbf{M_{e,y,i}} \mathbf{EM^{T}_{e,x,i}}$.

To simplify reading, we introduce the following notation: R^a, R^b, and R^t denoting the ray space created from the contents in A, B, T.

We can express the previous objective function as:

$$R^t \approx R^a \circ R^b \tag{4.18}$$

We can then create the following update rules:

$$\mathbf{A} \leftarrow \mathbf{A} \circ \frac{Y^a (R^t \circ R^b) X^a}{Y^a (R^a \circ R^b \circ R^b) X^a} \tag{4.19}$$

$$\mathbf{B} \leftarrow \mathbf{B} \circ \frac{Y^b (R^t \circ R^a) X^b}{Y^b (R^b \circ R^a \circ R^a) X^b} \tag{4.20}$$

4.6 Ray Casting Strategy

Until now, we have not concerned ourselves with how rays should be cast. A simple strategy for one viewpoint is one ray per pixel in the target content. This ensures maximal coverage of the target content. However, if the target content is of lower resolution, this might introduce aliasing, as pixels in the panels may be skipped over

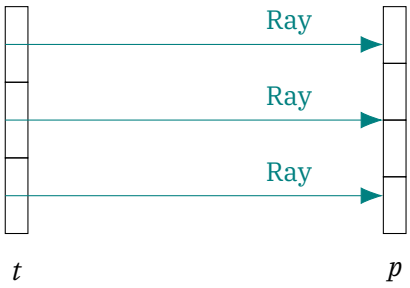


Figure 4.12: Potential source of ailising

Casting from observer to panels introduces the opposite problem for certain scenes. Some target content may never be sampled. Both approaches have rays that are easy to separate, as their origin is a point on a discretized plane.

Given that we have no guarantee that both panels have the same resolution, we believe it is more prudent to sample from the target. We will explore the possible ramifications of this decision in section 6.6.

4.6.1 Leveraging Sampling

Another strength of our separable approach is that it allows us to massively reduce sampling. Since the x and y dimensions are independent, we only need to sample each, not all their combinations. This property can be seen in Figure 4.10. a_5 is not sampled but is captured by our method. Taking our chosen ray casting strategy, this means we only need to sample the rays corresponding to the first row and first column of the target image. This brings our number of sampled rays from $O(n^2)$ to $O(n)$. Expanding this to multiple viewpoints, we get $O(n * o)$, where o is the number of viewpoints.

4.7 Dealing with void results

Our generation of content for the screens faces one last challenge: How to manage pixels that have not been sampled. Because of our mapping matrices, these pixels will always be assigned the value of 0. As developed in section 2.2, 0 is total occlusion. This results in panels that will occlude all pixels that are not directly sampled.

Replacing all 0 values with 1 is ill-advised, as it is possible some sampled pixels should be 0. Instead, we reuse the mapping tensors to determine if a pixel was sampled. If it wasn't, we change its value to 1.

For each pair of matrices corresponding to a viewpoint, we can construct a set of sampled rows and columns. Then, for each entry in the final image, we can check if there exists at least one matrix that sampled its row and column. If none does, then we know this pixel was not sampled and can set it to 1. An implementation can be found in section A.3

4.8 Motion Parallax

Now that we can account for multiple viewpoints and void results, we can model the eye box to achieve motion parallax. We sample multiple viewpoints close to each other using a 3D kernel. The exact kernel can be seen in Figure 4.13.

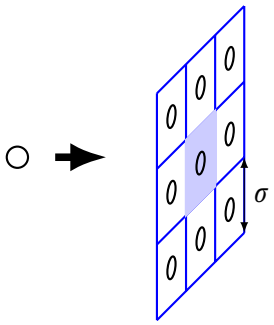


Figure 4.13: The kernel we use to capture the eyebox. We align the plane the kernel sits on to run parallel to the panels. σ controls the size granularity of the kernel.

Our kernel allows variations on the x and y axes, leaving us with 9 different positions for each eye.

Sampling each eye is independent, but the viewpoints of each eye are dependent.

5

Gallery

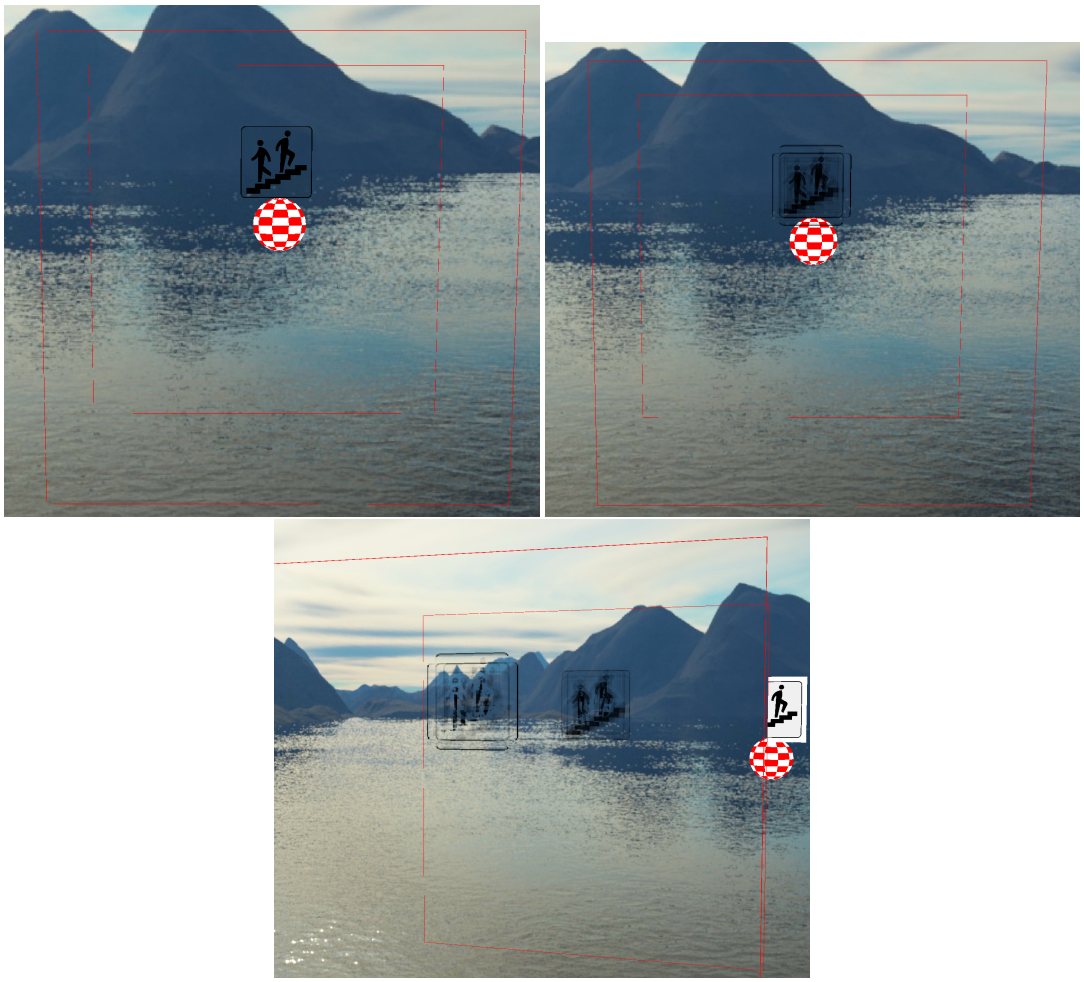


Figure 5.1: Our approach with an eyebox. The first image is the target texture. the second is our reconstructiong. The third has us rotate the camera to show the decomposition.

Figure 5.2: An icon seen from different positions within the eye box. The tucan is always visible, and moves similarly to the ball, a stand in for a real object. If not focused on, the tucan should seem blurry

6

Experiments

The following chapter presents a series of experiments we carried out to evaluate the separable approach developed in chapter 4. We are interested in run time, as well as image quality and potential tradeoffs between panel DPI and target resolution. This chapter begins by discussing our implementation details, as these have an impact on our results. We run an experiment comparing two versions of our method: tensor and matrix representation, demonstrating the superiority of the tensor representation. Huang, Chen, and Wetzstein's [21] stereoscopic approach is the closest in the field to what we have done, and serves as our next point of comparison. We compare their *The Light Field Stereoscope* against our method in sampling time and runtime. Our last experiment examines the tradeoffs between DPI and target resolution. We end this section by discussing potential threats to validity and the limitations of the conclusions that can be drawn from our results.

6.1 *The Light Field Stereoscope*

We reimplemented the method used in *The Light Field Stereoscope*. The update rule can be expressed as:

$$\vec{a} \leftarrow \vec{a} \circ \frac{\mathbf{\Phi}_a^T(\beta\vec{l} \circ (\mathbf{\Phi}_b\vec{b}))}{\mathbf{\Phi}_a^T((\mathbf{\Phi}_a\vec{a}) \circ (\mathbf{\Phi}_b\vec{b}) \circ (\mathbf{\Phi}_b\vec{b})) + \epsilon} \tag{6.1}$$

where $\vec{a}$ and $\vec{b}$ are vectorized forms of the display panels, and $\circ$ is the Hadmard or element-wise product. Φ is a sparse mapping matrix. A similar update rule can be made for b by swapping it with a. This update rule is almost identical to the one found in Equation 4.6 and Equation 4.7. The only difference is the $\beta\vec{l}$ term, which is the sampled light field. Instead of using the rays to report relations between elements, the stereoscopic approach immediately builds a target vector from the reported color values of the rays. This approach does not use a vectorized form of the target texture.

6.2 Implementation Details

Our implementation consists of the 4 components:

1. Ray Tracer Sampling: Written as WGSL compute shaders, this component samples the scene and produces the mapping matrices used by our approach and our re-creation of the stereoscope approach. We picked WGSL because it is portable and compiled. Though compilation introduces an additional step, the additional robustness gives us predictable behaviour across different drivers, which we will leverage to access different hardware and tools during our experiments. Additionally, the wgpu CPU interface can collect counters such as runtime timers and code traces.

2. Ray Tracer Rendering: Written as a WGSL vertex fragment shader, this component renders our scene. It can also take in the content produced by the factorizer

and render it. We use it to produce images and compute similarity between results. We picked wgpu due to its portability and ease of use.

3. Scene Control: Written in Rust, this component manages all scene details: the location of the panels, the target content, panel content, other objects, etc. We picked Rust because of its high-performance and easy concurrency. Primary builds on the cg math[1] crate for scene placement and the crevice[2] crate for serializing data to the std140 memory layout for GPU use. We use egui[3] and winit[4] for the UI and window management. Lastly, we run the shaders with wgpu[5].

4. Factorization: Written in Rust with the `faer`[6] crate, this component takes the various matrices and executes the algorithm. We picked Rust due to its performance and safety. We picked the `faer` crate after benchmarking it against other linear algebra crates. Additionally, `faer`'s support for sparse matrices made it an easy choice.

A simple diagram representation can be found at Figure 6.1.

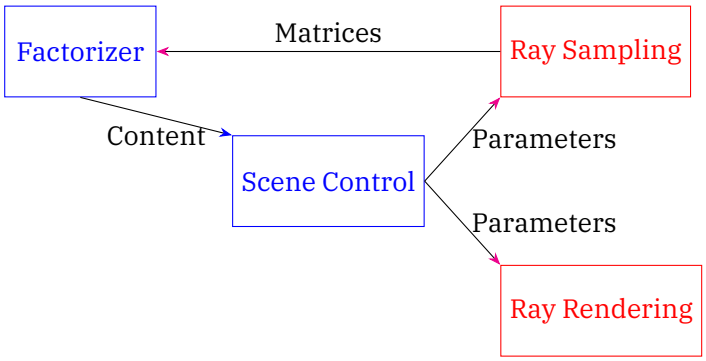


Figure 6.1: Layout of our Solution. Red is code on the GPU, blue on the CPU. Magenta Arrows represent this barrier being overcome

Communication between the GPU and CPU is done through a buffer. We ensure that every thread writes to a unique memory location in the buffer by flattening its unique x,y,z coordinates.

The separable sampler has a work group of size (64,2,1). We take the greatest dimension of the target texture and divide it by 64 to determine the size of the dispatch in the x dimension. The dispatch is always 2 dimensions in the z dimension (one for rows, one for columns) and the number of viewpoints in the y dimension.

By contrast, the stereo sampler has a work group of size (8, 8, 1). We dispatch (width / 8, height / 8, number of viewpoints) work groups.

We selected 64 and 8 because we are working with AMD graphics cards, which have a warp size of 64. An illustration can be seen in Figure 6.2

At this time, `faer` does not multithread on sparse matrix multiplication. As such, to make comparisons between methods fair, we elected to represent all mapping matrices as sparse matrices.

6.2.1 Ray tracing

We use ray tracing to sample scenes. We implemented the Möller–Trumbore intersection algorithm [6] for triangle ray intersection. A WGSL implementation can be found in section A.1. We use the barycentric coordinates to sample the textures that are mapped onto the triangles composing our target.

[1] https://docs.rs/cgmath/latest/cgmath/
[2] https://docs.rs/crevice/latest/crevice/
[3] https://docs.rs/egui/latest/egui/
[4] https://docs.rs/winit/latest/winit/
[5] https://wgpu.rs/
[6] https://docs.rs/faer/latest/faer/

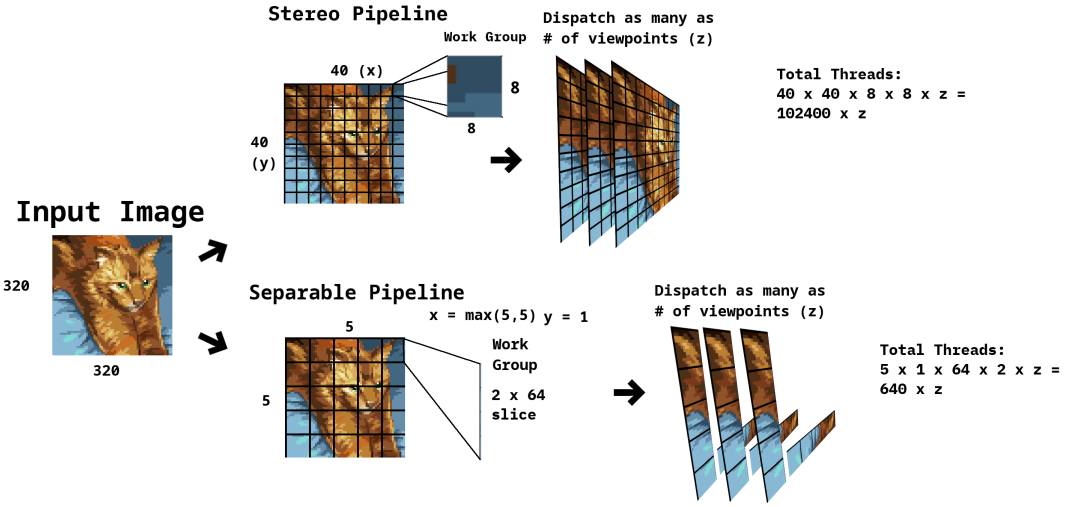


Figure 6.2: Our two pipelines. To make comparison easier, we have swapped the y and z dispatch on the separable pipeline.

6.2.2 How many iterations?

As developed in chapter 4, our approach implements an iterative update rule. To determine the number of iterations we will use for our experiments, we have computed the L2 norm of the target function over time. Figure 6.3 shows the L2 norm over 1000 iterations and Figure 6.4 shows the L2 norm over 5 iterations. *The Light Field Stereoscope* used only 1 update of the iteration rule, but there are still gains to be had.

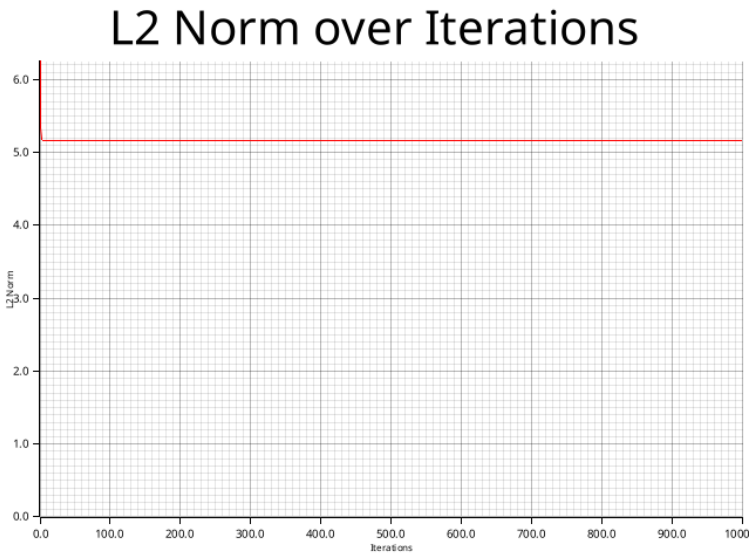


Figure 6.3: The L2 norm being minimized over 1000 iterations.

We concluded that 5 iterations were sufficient, as gains became minimal with more iterations.

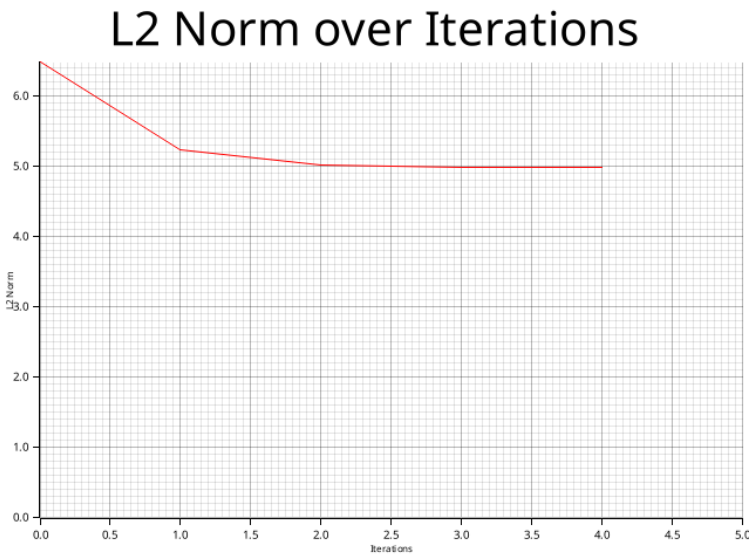


Figure 6.4: The L2 norm being minimized over 5 iterations.

6.3 Tensor Effect

In this section, we will compare the runtime of the separable tensor approach with its predecessor, the matrix representation. We will focus on panels at sizes 500 and 2000.

The benchmarks we wrote to compare run time were written in Criterion[7]. All were given 30 seconds of warm-up time and ran 200 times. We ran 5 iterations of the alternating least squares.

Benchmarks were run on Windows 11 Enterprise, with 32 GB of RAM, an AMD Ryzen 7 7800X3D 8-Core Processor at 4.20 GHz and an AMD Radeon RX7800 XT GPU with 16 GB of vRAM.

Panel Size	Target Size	Tensor	Matrix	Tensor Kernel	Matrix Kernel	Tensor 2 Kernel	Matrix 2 Kernel
500	256	0.279	0.403	0.279	0.404	0.684	2.500
	500	0.642	0.961	0.642	0.960	1.692	4.670
	100	1.032	4.685	1.032	4.680	2.194	21.400
	2000	2.934	21.400	2.834	21.400	5.972	89.192
2000	256	1.276	0.708	1.232	0.709	2.452	2.960
	500	1.782	1.400	1.792	1.400	3.572	5.570
	1000	2.312	5.900	2.313	5.600	4.515	23.140
	2000	4.744	23.200	4.754	23.200	9.334	92.500

Table 6.1: Runtime (s) for different representations. Tensor stands for the tensor method developed in Equation 4.18 while matrix stands for the naive matrix multiview discussed in Equation 4.14. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8. 2 Kernel is two different eyeboxes.

We can see in Table 6.1 that apart from the smallest target size, the tensor representation is significantly faster, up to 10 times faster for multiple viewpoints and a high resolution target texture. This is not surprising, as part of the reason we moved beyond the matrix representation was to avoid computing the entire ray space, which scales with the number of observers and target resolution. However, being able to quantify that difference and verify our assumptions is valuable.

6.4 Sampling Experiments

We measure the time taken by our ray casting shaders using wGPU `Timestamp Queries`. By placing a query before and after each shader call, we can calculate the time spent in the shader. For our experiment, we wrote a benchmark that warms up the cache by

[7] `https://bheisler.github.io/criterion.rs/book/`

running the shaders 100 times. Once the cache is warmed, we rerun the shader 100 times while recording the timestamps.

Experiments were run on Windows 11 Enterprise, with 32 GB of RAM, an AMD Ryzen 7 7800X3D 8-Core Processor at 4.20 GHz and an AMD Radeon RX7800 XT GPU with 16 GB of vRAM.

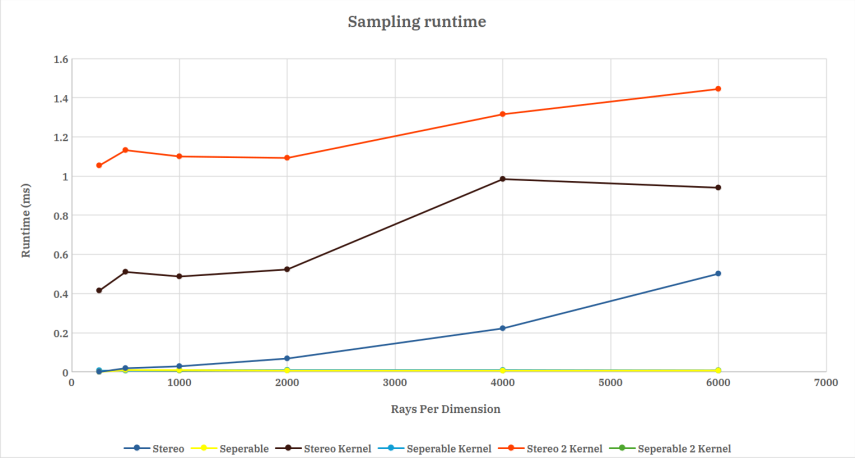


Figure 6.5: Time taken to sample as target texture becomes larger. Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8. 2 Kernel is two different eyeboxes. Note that all separable methods overlap as they take the same amount of time to sample.

	Time to Sample (ms)					
Scene	Stereo	Separable	Stereo Kernel	Separable Kernel	Stereo 2 Kernel	Separable 2 Kernel
256	0.01	0.01	0.42	0.01	1.05	0.01
500	0.02	0.01	0.51	0.01	1.13	0.01
1000	0.03	0.01	0.49	0.01	1.10	0.01
2000	0.07	0.01	0.52	0.01	1.09	0.01
4000	0.22	0.01	0.98	0.01	1.31	0.01
6000	0.50	0.01	0.94	0.01	1.44	0.01

Table 6.2: Ray Sampling table. Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8. 2 Kernel is two different eyeboxes.

We can clearly see the difference between the two approaches. Since the stereoscopic approach needs to run for every pixel in the target image, times the number of viewpoints, it grows exponentially. Meanwhile, the separable approach runs twice, once for the first column and once for the first row.

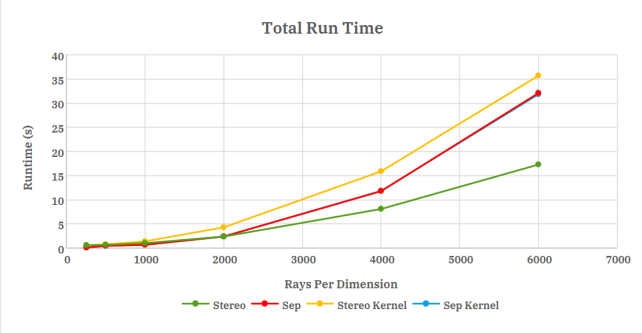
6.5 Factorization: Separable and Stereoscopic

We begin by looking at a relatively simple scene: two panels at 300 by 300 resolution, with multiple target images. Unlike our previous experiment, we start at this low panel resolution to explore beyond runtime. We will be measuring two different processes: build time and factorization time. The former is the time it takes to build the matrices from the GPU sampling, the later the time taken to solve 5 iterations of the update rule. We turn to runtime analysis to explain the differences between our approach and the stereoscopic one. We will later explore multiple combinations of panel and texture resolution. Then, we expand our comparison to consider multiple resolutions to draw greater conclusions about the entire problem space.

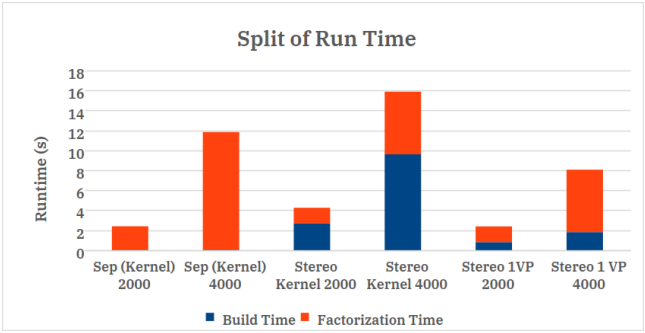
All benchmarks were run as described in section 6.3.

For larger target images, 2k and up, we ran into memory challenges when we sampled with 2 kernels for stereo. The vectorized form of the light field required more than

2 GB of VRAM, beyond the maximum size for a continuous buffer on our target hardware. As such, we removed those runs from consideration for direct comparison.



(a) Total Time Spent per approach. Note that both separable approaches overlap, with almost identical run times.



(b) Split of Time between building and factorization. Building is the time needed to build the mapping matrices from the data on the GPU. Factorization is the time spent solving the problem

Figure 6.6: Data for 300 by 300 panels

	Time to produce content (s)			
Scene	Stereo	Sep	Stereo Kernel	Sep Kernel
256	0.57	0.10	0.59	0.10
500	0.66	0.46	0.77	0.48
1000	1.02	0.67	1.40	0.70
2000	2.39	2.40	4.30	2.40
4000	8.10	11.8	15.89	11.84
6000	17.3	32.1	35.7	31.85

Table 6.3: Total Time Taken to produce content Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8.

Configuration	Build Time (s)	Factorization Time (s)
Sep Kernel 2000	0.035	2.36
Sep Kernel 4000	0.04	11.8
Sep 2000	0.035	2.36
Sep 4000	0.04	11.8
Stereo Kernel 2000	2.67	1.58
Stereo Kernel 4000	9.61	6.28
Stereo 2000	0.8	1.58
Stereo 4000	1.8	6.27

Table 6.4: Split between time taken to factorize matrices and time taken to build mapping matrices. The number after the name is the size of the texture for that particular scene. Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8.

There are two interesting questions that emerge from analyzing the source of runtime: Why does build time increase so drastically for the stereoscopic approach, and why is our factorization almost twice as slow?

The build time question is relatively easy to answer. Unlike our approach, the stereoscopic approach does not have a target texture. Instead, it builds the target light field as it samples. Every light ray takes 4 bytes to record, assuming 32-bit floating-point representations. The amount of rays cast to sample a scene is closely tied to the number of viewpoints, which the kernels greatly increase. As these increase, more and more data needs to be passed between the GPU and the CPU. For instance, we ran into problems for 2 kernels with 6000 pixels per dimension of the target image. At 6000 pixels per dimension, with 2 kernel, that's $6000 \times 6000 \times 9 \times 2 \times 4$ bytes, which is 2.60 GB. Going

down to 4000 with 1 kernel, that's 576 MB. That is still a considerable amount of data to unmap. The same configuration for the stereoscopic approach is 0.288 MB.

To analyze our factorization times, we must dig deeper. We turn to perf [46] and likwid[8] to give us greater insight. These tools are only available on Linux, and therefore the following experiments were run on a different machine: Framework 16, 16 × AMD Ryzen 7 7840HS w/ Radeon 780M Graphics, 32 GiB RAM, Arch Linux, kernel 6.15.9 We will be investigating the 2000 range. To avoid measuring GPU dispatches and unrelated instructions, we save the GPU data to disk. Then, in a headless version of the program, we read from disk, rebuild the mapping matrices, and then solve the system. We also include measurements for just loading the data.

Operation	Instructions	MFLOPs	Cache Access	Cache Miss	Branches	Branch Miss
Loading Data	6.7 G	7	3.9 T	2.45%	1.38 G	21.5%
Stereo	8.1 G	456	14.0 T	1.30%	1.49 G	17.4%
Separable	11.9 G	828	15.0 T	2.57%	2.10 G	8.5%

Table 6.5: Metrics Captured by likwid and perf. G for giga, T for tera.

From Table 6.5, we can see that the separable factorization gets executed as more instructions, but similar levels of cache access. One source of additional instructions is the on-the-fly light field generation. Our method multiplies the content on the target by two tensors to generate the light field for that given viewpoint. This introduces some additional computations. Additionally, at every step, we need to multiply the target content by two matrices instead of one. It is not surprising, then, that the number of instructions is twice as high.

6.5.1 Panel and Target Scaling

The question of how both approaches scale with panel size remains open. To investigate that, we conduct the same benchmarks as we did in section 6.5, but across wider scenes. Figure 6.7 displays all runs.

[8] https://github.com/RRZE-HPC/likwid

Figure 6.7: Runtime with all parameters Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8.

Target Size	Sep	Stereo	Kernel Sep	Kernel Stereo	2 Kernel Sep	2 Kernel Stereo
256	0.087	0.549	0.087	0.58	0.142	0.641
500	0.419	0.633	0.423	0.745	0.92	0.973
1000	0.58	0.983	0.58	1.434	1.104	2.26
2000	2.204	2.304	2.204	4.17	4.232	7.95

Table 6.6: Runtime (s) Data for panels 250 by 250. Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8. 2 Kernel is two different eyeboxes.

Target Size	Sep	Stereo	Kernel Sep	Kernel Stereo	2 Kernel Sep	2 Kernel Stereo
256	0.279	0.589	0.279	0.589	0.684	0.676
500	0.642	0.755	0.642	0.753	1.692	1.012
1000	1.032	1.441	1.032	1.431	2.194	2.375
2000	2.934	4.19	2.834	4.18	5.972	8

Table 6.7: Runtime (s) Data for panels 500 by 500. Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8. 2 Kernel is two different eyeboxes.

Target Size	Sep	Stereo	Kernel Sep	Kernel Stereo	2 Kernel Sep	2 Kernel Stereo
256	0.367	0.619	0.369	0.619	0.683	0.674
500	0.651	0.779	0.825	0.78	1.699	1.014
1000	0.951	1.472	1.132	1.461	2.192	2.375
2000	3.035	4.21	3.034	4.21	5.974	7.99

Table 6.8: Runtime (s) Data for panels 1000 by 1000. Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8. 2 Kernel is two different eyeboxes.

Target Size	Sep	Stereo	Kernel Sep	Kernel Stereo	2 Kernel Sep	2 Kernel Stereo
256	1.276	0.735	1.232	0.738	2.452	0.794
500	1.782	0.907	1.792	0.909	3.572	1.134
1000	2.312	1.591	2.313	1.59	4.515	2.486
2000	4.744	4.34	4.754	4.35	9.334	8.16

Table 6.9: Runtime (s) Data for panels 2000 by 2000. Stereo is *The Light Field Stereoscope*, Separable is our method. Kernel stands for a single eyebox represented by our kernel approach developed in section 4.8. 2 Kernel is two different eyeboxes.

6.5.2 Analysis

From the data, we can observe that the separable approach performs better than the stereoscopic approach. However, in the highest panel, highest target resolution, the stereoscopic approach performs better. Additionally, it would seem that the cost of a second eye box is vastly greater than the first. For the separable approach, the first eyebox is free. Upon investigating the assembly using `cargo-show-asm`[9], we can rule out SIMD being responsible, as no instructions are SIMD instructions. Neither implementation uses SIMD instructions. It is likely then that some increase in viewpoints pushes us over a cache line.

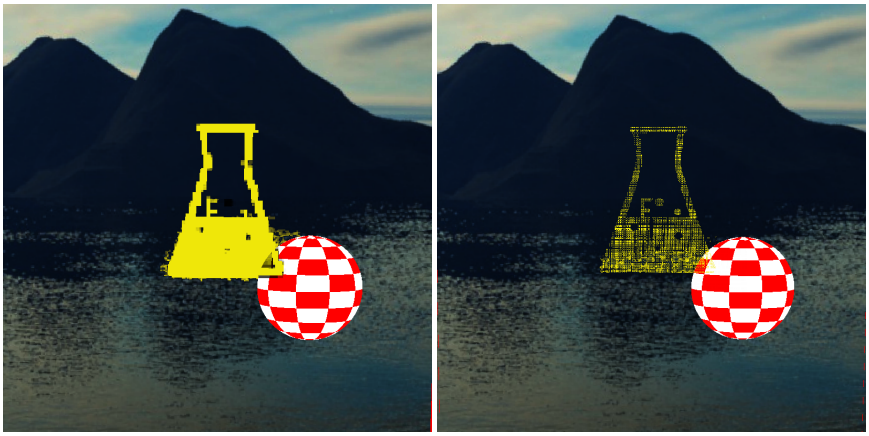
When observing the totality of the data, it becomes evident that the separable approach scales according to the target size and the panel size. Meanwhile, the stereoscopic approach is more susceptible to the target size, but unaffected by the panel size. This raises an interesting question, which we will address in the next section: what is the ideal relationship between target size and panel size? We have already discussed in section 4.6 the trade-offs between ray-casting strategies. Relatively low-resolution

[9] `https://github.com/pacak/cargo-show-asm`

targets will lead to aliasing, while low-DPI panels will have an impact on our reconstruction.

6.6 Panel DPI and Target Resolution

This section deals with the trade-off between the DPI of the panels and the resolution of the target. To investigate this, we compare the output of our method with the image produced in the virtual world, using the structural similarity index measure (SSIM). To make this a fair comparison, we use the same target image at different resolutions. We use a target image that is not binary in its content (not just black and transparent). We use a single viewpoint.



(a) Target is 2000 by 2000, panel is 250 by 250 **(b)** Target is 250 by 250, panel is 2000 by 2000

Figure 6.8: The different types of aliasing. Output has been edited to produce yellow to highlight the difference. The Target texture is at a distance of 5. High-resolution target with low-resolution panels results in low-resolution images, with pronounced pixels. Low-resolution target with high-resolution panels results in undersampling, with rows and columns of pixels that are transparent. In this setup, transparent pixels show the black of the mountain.

		Target					
		250	500	1000	2000	4000	6000
Panel	250	0.009945	0.009832	0.009774	0.009770	0.009787	0.009792
	500	0.008222	0.008314	0.008248	0.008186	0.008183	0.008176
	1000	0.010232	0.007173	0.007490	0.007422	0.007419	0.007404
	2000	0.010899	0.009875	0.006779	0.007256	0.007231	0.007206
	4000	0.011090	0.011102	0.010073	0.006473	0.007054	0.007056
	6000	0.011125	0.011024	0.010889	0.008086	0.007034	0.007014

Table 6.10: SSIM ×100 for different configurations. Lower is better. Target is 1 away from panels.

		Target					
		250	500	1000	2000	4000	6000
Panel	250	0.002968	0.002989	0.002982	0.002960	0.002967	0.002962
	500	0.002539	0.002606	0.002545	0.002539	0.002532	0.002532
	1000	0.002421	0.002398	0.002354	0.002342	0.002337	0.002336
	2000	0.002446	0.002341	0.002281	0.002276	0.002269	0.002268
	4000	0.002505	0.002396	0.002274	0.002253	0.002249	0.002247
	6000	0.002504	0.002486	0.002338	0.002244	0.002240	0.002237

Table 6.11: SSIM×100 for different configurations. Lower is better. Target is 5 away from panels.

		Target					
		250	500	1000	2000	4000	6000
Panel	250	0.001002	0.000999	0.000989	0.000987	0.000986	0.000987
	500	0.000921	0.000921	0.000916	0.000915	0.000915	0.000915
	1000	0.000878	0.000880	0.000877	0.000872	0.000871	0.000871
	2000	0.000856	0.000848	0.000841	0.000838	0.000838	0.000838
	4000	0.000849	0.000837	0.000823	0.000822	0.000820	0.000820
	6000	0.000877	0.000833	0.000823	0.000818	0.000818	0.000817

Table 6.12: SSIM ×100 for different configurations. Lower is better. Target is 10 away from panels.

As content is further away, the SSIM difference is smaller. This makes sense, as we would perceive a smaller image.

Table 6.10 shows that more is not necessarily strictly optimal. The ideal combination is a panel 4000 by 4000 and a target that is 2000 by 2000.

As distance increases, the higher resolution and DPI seem to matter more and more, but it is difficult to say if this is a break from the trends seen in table 6.10 or if we have seen a shift to the left.

All tables also show that if you fix the target, an increase in Panel does not necessarily translate into an increase in score. Given that our target controls the number of rays cast, this makes sense. Apart from table 6.10, increasing the target size generally increases the score. Additionally, the comparing the 1000 Panel range in table 6.11 and table 6.12, it would appear that the further a texture is placed, the more the lower panels become inadequate, no matter the compensation on the target size.

This makes intuitive sense: the further out the target texture is, the smaller we expect that texture to look on our panels. More sampling cannot overcome the limitation of low resolution.

6.7 Threats to Validity

The first threat to validity comes from multithreading. Vector-matrix multiplication is not necessarily faster than matrix-matrix multiplication [29], and sparse-matrix multiplication is still an area of ongoing research [43]. An implementation of the factorization that does multithreading on the sparse-matrix multiplications may have wildly different results. However, any multithreading approach would only be able to parallelize the factorization time, not the communication time. We classify the separable approach as embarrassingly parallel, with less than 1% being spent in communications. The stereoscopic approach, on the other hand, spends between 30 to 60% of its run time in factorization. We will take the most generous assumption and extend 60% as the general case. Let's assume both sparse-matrix multiplication and vector sparse-matrix multiplication benefit from the same speed-up. Let us call the theoretical speed-up of the factorization f. From that, using Amdahl's law [3], [4] we can get

the latency speed up:

$$S_{\text{sep}} = \frac{1}{1 - 0.99 + \frac{0.99}{f}} \approx \lim_{f \to \infty} \frac{f}{0.01f + 0.99} = 100 \tag{6.2}$$

$$S_{\text{stereo}} = \frac{1}{1 - 0.6 + \frac{0.6}{f}} \approx \lim_{f \to \infty} \frac{f}{0.4f + 0.6} = 2.5 \tag{6.3}$$

We would therefore expect a multithreaded approach to greatly benefit our method, assuming the speed-up is similar for both operations.

The second threat to validity comes from GPU acceleration. The main reason the stereoscopic approach is slower is due to the price of communication between the GPU and CPU and the ever-increasing memory footprint of that approach. In theory, GPU acceleration of the factorization could completely overcome this challenge, as no buffer would have to be transferred. However, there are some additional factors to consider. The memory footprint of the stereoscopic approach is much greater, and this resource is even more scarce on the GPU. This would still not overcome the buffer size challenge we ran into that limited the scope of experiments. Additionally, the three dimensions of the tensors in the separable approach might map more effectively to work group dispatches, leading to better warp usage. Nonetheless, this certainly is an open question for future researchers.

6.8 Chapter Summary

In this chapter, we have discussed our implementation of our content generation method. Next, we compared our method to a variant that would use matrix representation instead of tensors. We observed that apart from the lowest resolution, the tensor approach eclipsed the matrix approach, in some cases by one order of magnitude.

We then examined it against the closest work in the literature. Both were benchmarked across panel sizes and target resolution. We found that apart from the highest target resolution and panel dpi, our method generated content faster. We identified the GPU-CPU communication as being the main bottleneck for the stereoscopic approach. It factorizes faster than ours, but needs to push significantly more data across. Scrutinizing this factorization time, we confirmed that our method compiles to more instructions and more floating-point operations. We also investigated caching and branching, finding both methods to be in similar orders of magnitude.

Following that, we investigated whether a higher DPI invariably leads to better perceived images. We found that this is impacted by the distance between the panels and the texture. However, evidence suggests that there is an ideal middle ground.

Lastly, we discussed potential threats to validity coming from different implementations of our foundational building blocks: sparse matrix multiplication. Any change to this might significantly impact our results.

7

Discussion and Conclusion

We begin this chapter by discussing our research, and interesting potential applications. Following that, we examine the limitations, from potential flaws in our modelling to constraints we did not consider. Next, we present the future work, both extensions and alternatives to our approach. We conclude the thesis.

7.1 Discussion

In this thesis, we have proposed a decomposition for near eye displays that supports correct focus cues. This decomposition started with a naive approach, and gradually became more focused to our problem space. We tackled new challenges, such as the void results, not seen in any work in the literature.

The greatest gain from our approach was the sampling time, being 2 orders of magnitude smaller, and being invariant to the number of observes. Looking at the render time, we achieve lower times than those found in the literature. This was possible due to the smaller footprint of our method. Our method experienced no noticeable communication overhead. We also explored and ruled out other hypotheses, such as a hidden SIMD implementation, or greater cache access.

Following that, we explored the relationship between reconstruction quality, panel DPI and target resolution. We observed that reconstruction score depended greatly on the scene, and that greater target resolution or panel resolution did not necessarily translate to a major score improvement.

We believe there are some interesting use cases that emerge from our findings.

2D Animated Content

One interesting implication of our method is the on-the-fly light field generation. Unlike previous approaches, assuming our observers do not move, we can reuse the same mapping matrices for different content, by swapping out the content matrix. This means we can avoid the sampling step for 2D animated content, such as a digital clock!

In our experiments, we saw that the sampling time played almost no role in the total run time. However, as we will discuss later, different implementation approaches, such as multithreading or GPU acceleration, might change this. The scene sampling is already GPU accelerated.

Embedded System and Server-Client Solution

An appealing potential application of our system is a client-server system. Given our much lighter data footprint for sampling, we can envision a simple embedded system that captures sensor data and samples the virtual world. Subsequently, it sends this data over to a more powerful machine for image generation and gets back two images to display on the panels. Given current improvements in Wi-Fi capabilities [44], our data load of kilobytes should have minimal latency. The images being sent back would scale with the size of the panels, but it is likely not the bottleneck of this system. We would expect the computation to be the most intense step, as such a system should

have latency below 50 ms to avoid cybersickness, ideally below 20 ms [38]. Such a system would likely benefit from greater battery life, having removed two expensive components: backlight and computation.

Additive Approach

In chapter 3, we discussed a different approach, additive light fields. We argued that additive approaches do not solve the autonomy challenge we laid out in chapter 1 The question remains: can our approach be adapted to an additive set up?

The main contribution we made was the leveraging the separability of rays to capture the problem space in a more compact manner. We capture the observers as a separate dimension, allowing us to turn a matrix multiplication into an element wise summation.

There are some elements used by additive approaches that may pose a challenge. The first is the use of lenses and beam splitters [27]. These risk breaking the underlying geometry we build on to assume ray separability . Next, some additive approaches use of tomography for scene sampling [27]. This would simply not work for 2D content.

We believe that there may be elements that inspire future additive works, notably including the target texture in the constraint formulation. However, there is a limit to how much can be translated. Many additive approaches do not have to consider the same kind of content decomposition across panels. The main challenge is to configure their optical elements to correctly blur the projected content when the user focuses away. Narian et al. [24] does consider a panel decomposition, but works with the problem in the frequency domain.

7.2 Limitations

Only three flatlands

One great limitation is that there is only 1 target texture. It is evident that AR applications are interested in rendering more than 1 target. We believe that there are methods to overcome this, but would require extending our approach. This likely involves reformulating the problem to include an arbitrary number of targets. We suspect an extension to the tensor representation could handle this, by adding a fourth dimension to the mapping tensors. This would allow a similar approach as multiple observers, preventing the ballooning of the ray space.

However, we would worry about this having a major impact on computation. Each additional target risks multiplying the number of viewpoints. As we observed in Figure 6.7, the additional view points did increase run time.

Transparency of screens

We have assumed that the screens are perfectly transparent. Presently, they are not. Transparency has gone up over the years, as research pushes on. However, whether it is even possible to achieve screens with 100% transmittance is unclear. A more complete representation may be capable of factoring in the transmittance rate into the objective function.

No hardware prototype

We have not presented any hardware prototype, and therefore have avoided many of the challenges that a real implementation would face. Refraction and reflection on the screens themselves may introduce significant noise. Additionally, we do not consider the optical elements that are necessary to construct such a light field display.

7.3 Future Work

There are many ways to extend this work, both at a theoretical level and implementation wise.

Different Formulations
There may be a more effective approach to the constraint solving. One of the most expensive operations is the need to transform to ray space and back to content space for all content. If a formulation can be derived that solves in ray space, and transforms back to content space at the end, this would save many computations. That is to say, our update rule would act on $\mathbf{M_{a,y}AM_{a,x}^T}$ instead of $\mathbf{A}$, only transforming back at the end.

Different Eyebox and Weighted views
One feature of our method we did not utilize is assigning weights to the viewpoints. A more developed model of the eye could use a more sophisticated kernel, which would weigh viewpoints differently. This quick addition could create more convincing decompositions.

Continuous Flatlands
Moving away from reasoning purely in the discrete space may allow us to more accurately capture the light field generated by the target texture. By first exploring the problem in the continuous space, there may be interesting properties that can be exploited for faster and better content generation. This would be a great departure from our current methods.

SIMD and Multithreading
As discussed in section 6.7, multithreading may unlock great gains. In addition to that, there is work on SIMD portability in Rust,[1], currently available on the nightly release, which `faer` has support for. SIMD is of particular interest given the number of elementwise operations we perform. This may unlock even more gains on top of those gained by multithreading.

GPU Acceleration
GPU acceleration is necessary if we wish to apply this method to a real system. As of now, even if we decrease the iterations to 1, as some previous research has [21], we would still be above the 50 ms for latency. GPU acceleration promises great gains, but correctly adapting our approach is not trivial.

7.4 Conclusion
By considering the specifics of our needs, we were able to narrow down the type of content that would be most useful. This limited us to 2D texture. Leveraging this required a more complex approach than had been done previously, but resulted in gains. We invite others to walk through *the three flatlands*, and understand it's all a matter of perspective.

[1] `https://github.com/rust-lang/rust/issues/86656`

8

Contemplating Flatlands: A Retrospective

This chapter is my retrospective on these last 6 months, what I did correctly, and what I think I can do better on. I say “I” and not “we” because as Umberto Eco [20] laid it out, “we” includes the reader, whereas this chapter is self-reflection. It should not be read as part of my research but rather as my musings on process, on being a researcher, and on what I have learned. I include it for my future self and for those who might be about to begin their thesis.

Looking back, my greatest drive is also part of my greatest weakness. My desire to move towards implementation and concreteness, wanting to avoid sitting on a problem for excessive amounts of time, motivated me to start my research quickly. It pushed me to ask for help from friends, from my supervisor, or from Dr. Romeir. Research can be a solitary activity, but that does not mean others should not be involved. This is also what drove me to build systems that allowed me to test on mass, not just automating my data collection but allowing me to quickly spin up new types of experiments that would demonstrate different properties.

In my rush towards the tangible, I neglected a solid foundation. It took me a long time to develop a proper understanding of the problem space, and the exact research question was determined in late June, 4 months into the research. Spending more time establishing that foundation could have helped me avoid many of the dead ends I ran into. Originally, my implementation was done in OpenGL, a technology I was familiar with. However, as the needs of the research evolved, I ran into limitations that forced me to change to a different framework. Had I spent more time designing my solution and thinking about which elements I might need, I might have picked wgpu from the start and avoided that morass.

Lastly, the importance of working out the orders of magnitude ahead of time cannot be overstated. Napkin math exposed many of the false paths I took and should have been part of my process from the beginning. It is easy to lose the sense of things when the orders of magnitude leave the 100s and the dimensions grow beyond the second.

A

Appendix

A.1 Moller-Trumbore Intersection

Here is an implementation of the Moller-Trumbore code in wgsl, as a computer shader.

```
fn intersection_panel(ray: Ray, panel: Panel) -> PanelIntersection {
    var hit = false;
    var a = panel.quad.a;
    var b = panel.quad.b;
    var c = panel.quad.c;
    var bary_coords = vec3f(0, 0, 0);

    var e1 = b - a;
    var e2 = c - a;
    var rey_cross_e2 = cross(e2,(ray).direction);
    var det = dot(rey_cross_e2, e1);

    if det > -eps && det < eps {
        return PanelIntersection(hit, border, pixel_coords);
    }
    var inv_det = 1.0 / det;
    var s = (ray).origin - a;
    var u = inv_det * dot(rey_cross_e2, s);

    if (u < 0.0 && abs(u) > eps) || (u > 1.0 && abs(u - 1.0) > eps) {
        return PanelIntersection(hit,bary_coords);
    }
    var s_cross_e1 = cross(e1, s);
    var v = inv_det * dot(s_cross_e1,(ray).direction);
    var w = 1.0 - v - u;

    if v < 0.0 || u + v > 1.0 {
        return PanelIntersection(hit, bary_coords);
    }

    var t = inv_det * dot(s_cross_e1, e2);

    if t > eps {
        hit = true;
        let bary_coords = vec3f(u, v, w);
        return PanelIntersection(hit, bary_coords);
        }
    else {
        return PanelIntersection(hit, bary_coords);
```

```
    }
```

A.2 Update rule Implementation

The following code snippet shows the implemtnation of the update rules as developed in 4.19 and 4.20. This snippet is only the update for the panel A.

```
for view_point in 0..number_of_view_points as usize {
    let m_a_x = matrices.a.x.matrix[view_point].as_ref();
    let m_a_y = matrices.a.y.matrix[view_point].as_ref();

    let m_b_x = matrices.b.x.matrix[view_point].as_ref();
    let m_b_y = matrices.b.y.matrix[view_point].as_ref();

    let m_t_x = matrices.t.x.matrix[view_point].as_ref();
    let m_t_y = matrices.t.y.matrix[view_point].as_ref();

    let R_t = (m_t_y * &c_t) * m_t_x.transpose();
    let R_b = m_b_y * &c_b * m_b_x.transpose();
    let R_a = m_a_y * &c_a * m_a_x.transpose();

    zip!(&mut upper, &R_b, &R_t).for_each(
        |unzip!(upper, b, t)| {
            *upper = *b * *t;
        },
    );

    zip!(&mut lower, &R_b, &R_a).for_each(
        |unzip!(lower, b, a)| {
            *lower = *a * *b * *b;
        },
    );

    numerator_a += m_a_y.transpose() * &upper * m_a_x;
    denominator_a += m_a_y.transpose() * &lower * m_a_x;
}

zip!(&mut c_a, &mut numerator_a, &mut denominator_a).for_each(
    |unzip!(c_a, n, d)| {
        *c_a = 1.0_f32.min(*c_a * *n / (*d + 0.0000001f32));
        *n = 0.0;
        *d = 0.0;
    },
);
}
```

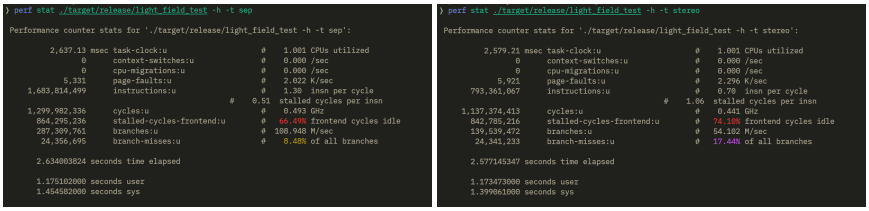
A.3 Filtering

```
pub fn filter_zeroes(
    mat: &mut Mat<f32, usize, usize>,
    mapping_mat: &CompleteMapping)
{
    let x_filter = filter_combine(&mapping_mat.x);
    let y_filter = filter_combine(&mapping_mat.y);
    let filters: Vec<(HashSet<usize>, HashSet<usize>)>
    = zip(x_filter, y_filter).collect();
```

```
    for (row, x) in mat.row_iter_mut().enumerate() {
        for (column, y) in x.iter_mut().enumerate() {
            if !filters
                .iter()
                .any(|set| set.1.contains(&row) && set.0.contains(&column))
            {
                *y = 1.0;
            }
        }
    }
}
fn filter_combine(mapping: &MappingMatrix) -> Vec<HashSet<usize>> {
    mapping.matrix.iter().map(active_columns).collect()
}
fn active_columns(mat: &SparseColMat<u32, f32>) -> HashSet<usize> {
    let pntr = mat.col_ptr();
    let mut set = HashSet::new();
    for column in 0..mat.ncols() {
        if pntr[column + 1] != pntr[column] {
            set.insert(column);
        }
    }
    set
}
```

A.4 Perf and Likwid

```
perf stat ./target/release/light_field_test -h -t sep;
perf stat ./target/release/light_field_test -h -t stereo;
```

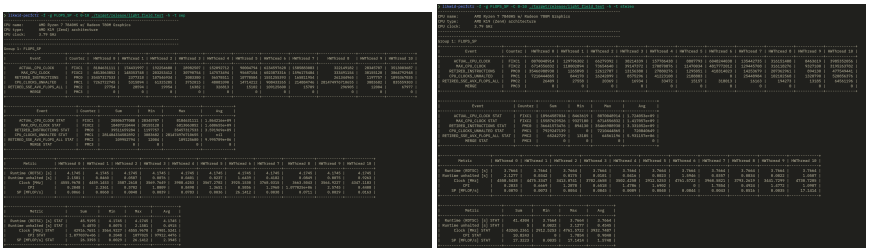


(a) Output of Perf for Seperable approach **(b)** Output of Perf for the Stereo Approach

Figure A.1: Output from both perf runs

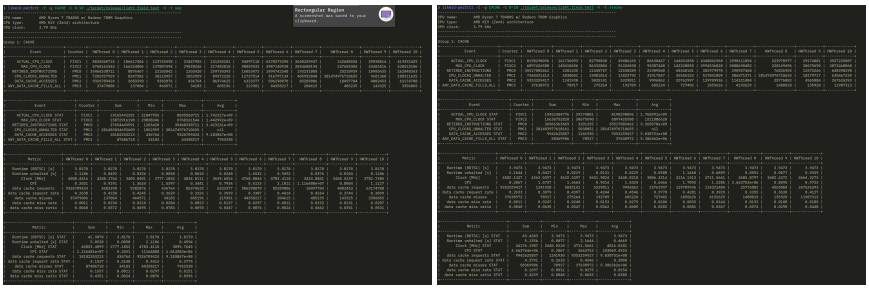
```
likwid-perfctr -g FLOPS_SP -C 0-10 ./target/release/light_field_test -h -t sep;
likwid-perfctr -g FLOPS_SP -C 0-10 ./target/release/light_field_test -h -t stereo;
```

```
likwid-perfctr -g CACHE -C 0-10 ./target/release/light_field_test -h -t sep;
likwid-perfctr -g CACHE -C 0-10 ./target/release/light_field_test -h -t stereo;
```



(a) Output of Perf for Seperable approach **(b)** Output of Perf for the Stereo Approach

Figure A.2: Output from both likwid FPSP runs



(a) Output of Perf for Seperable approach, **(b)** Output of Perf for the Stereo Approach

Figure A.3: Output from likwid Cache runs

Bibliography

[1] C. Wheatstone, "Xviii. contributions to the physiology of vision.—part the first. on some remarkable, and hitherto unobserved, phenomena of binocular vision," *Philosophical transactions of the Royal Society of London*, no. 128, pp. 371–394, 1838 (cit. on p. 4).

[2] F. E. Ives, *Us patent 725567a, parallax stereogram and process of making same.* 1902 (cit. on p. 5).

[3] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring), Atlantic City, New Jersey: Association for Computing Machinery, 1967, pp. 483–485, ISBN: 9781450378956. DOI: 10.1145/1465482.1465560. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/1465482.1465560 (cit. on p. 32).

[4] D. P. Rodgers, "Improvements in multiprocessor system design," *SIGARCH Comput. Archit. News*, vol. 13, no. 3, pp. 225–231, Jun. 1985, ISSN: 0163-5964. DOI: 10.1145/327070.327215. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/327070.327215 (cit. on p. 32).

[5] M. Oren and S. K. Nayar, "Generalization of the lambertian model and implications for machine vision," *International Journal of Computer Vision*, vol. 14, no. 3, pp. 227–251, 1995 (cit. on p. 5).

[6] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21–28, 1997. DOI: 10.1080/10867651.1997.10487468. eprint: https://doi.org/10.1080/10867651.1997.10487468. [Online]. Available: https://doi.org/10.1080/10867651.1997.10487468 (cit. on p. 23).

[7] K. Kiyokawa, Y. Kurata, and H. Ohno, "An optical see-through display for mutual occlusion of real and virtual environments," in *Proceedings IEEE and ACM International Symposium on Augmented Reality (ISAR 2000)*, IEEE, 2000, pp. 60–67 (cit. on p. 1).

[8] D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., vol. 13, MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf (cit. on pp. 10, 13).

[9] T. Sielhorst, T. Obst, R. Burgkart, R. Riener, and N. Navab, "An augmented reality delivery simulator for medical training," in *International workshop on augmented environments for medical imaging-MICCAI Satellite Workshop*, vol. 141, 2004, pp. 11–20 (cit. on p. 1).

[10] V. Blondel, N.-D. Ho, and P. Van Dooren, "Weighted nonnegative matrix factorization and face feature extraction," *Image and Vision Computing - IVC*, Jan. 2007 (cit. on p. 14).

[11] D. Lanman, M. Hirsch, Y. Kim, and R. Raskar, "Content-adaptive parallax barriers: Optimizing dual-layer 3d displays using low-rank light field factorization," *ACM Trans. Graph.*, vol. 29, no. 6, Dec. 2010, ISSN: 0730-0301. DOI: 10.1145/1882261.1866164. [Online]. Available: https://doi.org/10.1145/1882261.1866164 (cit. on p. 10).

[12] L. Wilcox and J. Harris, "Fundamentals of stereopsis," in *Encyclopedia of the Eye*, D. A. Dartt, Ed., Oxford: Academic Press, 2010, pp. 164–171, ISBN: 978-0-12-374203-2. DOI: https://doi.org/10.1016/B978-0-12-374203-2.00237-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123742032002372 (cit. on p. 4).

[13] N. S. Holliman, N. A. Dodgson, G. E. Favalora, and L. Pockett, "Three-dimensional displays: A review and applications analysis," *IEEE transactions on Broadcasting*, vol. 57, no. 2, pp. 362–371, 2011 (cit. on p. 5).

[14] D. Lanman, G. Wetzstein, M. Hirsch, W. Heidrich, and R. Raskar, "Polarization fields: Dynamic light field display using multi-layer lcds," in *Proceedings of the 2011 SIGGRAPH Asia Conference*, 2011, pp. 1–10 (cit. on p. 7).

[15] G. Wetzstein, D. Lanman, W. Heidrich, and R. Raskar, "Layered 3d: Tomographic image synthesis for attenuation-based light field and high dynamic range displays," *ACM Trans. Graph.*, vol. 30, no. 4, Jul. 2011, ISSN: 0730-0301. DOI: 10.1145/2010324.1964990. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/2010324.1964990 (cit. on pp. 7, 8).

[16] G. Wetzstein, D. R. Lanman, M. W. Hirsch, and R. Raskar, "Tensor displays: Compressive light field synthesis using multilayer displays with directional backlighting," 2012 (cit. on p. 7).

[17] A. Cirulis and E. Ginters, "Augmented reality in logistics," *Procedia Computer Science*, vol. 26, pp. 14–20, 2013 (cit. on p. 1).

[18] A. Maimone and H. Fuchs, "Computational augmented reality eyeglasses," in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2013, pp. 29–38. DOI: 10.1109/ISMAR.2013.6671761 (cit. on pp. 1, 7).

[19] Qwertyus, *Nmf*, [Online; accessed August 15, 2025], 2013. [Online]. Available: https://commons.wikimedia.org/wiki/File:NMF.png (cit. on p. 11).

[20] U. Eco, *How to write a thesis*. MIT Press, 2015 (cit. on p. 37).

[21] F.-C. Huang, D. P. Luebke, and G. Wetzstein, "The light field stereoscope.," in *SIGGRAPH emerging technologies*, 2015, pp. 24–1 (cit. on pp. 1, 2, 7, 22, 36).

[22] S.-K. Kim, K.-H. Yoon, S. K. Yoon, and H. Ju, "Parallax barrier engineering for image quality improvement in an autostereoscopic 3d display," *Optics express*, vol. 23, no. 10, pp. 13 230–13 244, 2015 (cit. on pp. 5, 11).

[23] G. Kramida, "Resolving the vergence-accommodation conflict in head-mounted displays," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 7, pp. 1912–1931, 2015 (cit. on p. 5).

[24] R. Narain, R. A. Albert, A. Bulbul, G. J. Ward, M. S. Banks, and J. F. O'Brien, "Optimal presentation of imagery with focus cues on multi-plane displays," *ACM Trans. Graph.*, vol. 34, no. 4, Jul. 2015, ISSN: 0730-0301. DOI: 10.1145/2766909. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/2766909 (cit. on pp. 8, 35).

[25] C.-W. Su, C.-C. Liao, and M.-Y. Chen, "Color transparent display using polymer-dispersed liquid crystal," *Journal of Display Technology*, vol. 12, no. 1, pp. 31–34, 2015 (cit. on p. 6).

[26] C. Anthes, R. J. García-Hernández, M. Wiedemann, and D. Kranzlmüller, "State of the art of virtual reality technology," in *2016 IEEE Aerospace Conference*, 2016, pp. 1–19. DOI: 10.1109/AERO.2016.7500674 (cit. on p. 5).

[27] S. Lee, C. Jang, S. Moon, J. Cho, and B. Lee, "Additive light field displays: Realization of augmented reality with holographic optical elements," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–13, 2016 (cit. on pp. 8, 35).

[28] C. Jang, K. Bang, S. Moon, J. Kim, S. Lee, and B. Lee, "Retinal 3d: Augmented reality near-eye display via pupil-tracked light field projection on retina," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 1–13, 2017 (cit. on p. 8).

[29] S. Kurt, V. Thumma, C. Hong, A. Sukumaran Rajam, and P. Sadayappan, "Characterization of data movement requirements for sparse matrix computations on gpus," Dec. 2017, pp. 283–293. DOI: 10.1109/HiPC.2017.00040 (cit. on p. 32).

[30] M. Park and H.-J. Choi, "A three-dimensional transparent display with enhanced transmittance and resolution using an active parallax barrier with see-through areas on an lcd panel," *Current Optics and Photonics*, vol. 1, no. 2, pp. 95–100, 2017 (cit. on p. 1).

[31] Y.-H. Tsai, Y.-L. Wu, W.-T. Liou, Y.-R. Kung, Y.-H. Huang, and K.-C. Lee, "P-202: A flexible transparent oled display with flexuptm technology," in *SID Symposium Digest of Technical Papers*, Wiley Online Library, vol. 48, 2017, pp. 2021–2024 (cit. on pp. 1, 6).

[32] Kuhnmi, *Airplane landing at zurich airport*, [Online; accessed August 27, 2025], 2018. [Online]. Available: https://commons.wikimedia.org/wiki/File:Airplane_landing_at_zurich_airport.jpg (cit. on p. 3).

[33] A. Basu, "A brief chronology of virtual reality," *arXiv preprint arXiv:1911.09605*, 2019 (cit. on p. 5).

[34] Y. Wang, G. Zhai, S. Chen, X. Min, Z. Gao, and X. Song, "Assessment of eye fatigue caused by head-mounted displays using eye-tracking," *Biomedical engineering online*, vol. 18, pp. 1–19, 2019 (cit. on p. 5).

[35] S. Bagassi, F. De Crescenzio, S. Piastra, *et al.*, "Human-in-the-loop evaluation of an augmented reality based interface for the airport control tower," *Computers in Industry*, vol. 123, p. 103 291, 2020 (cit. on p. 1).

[36] Z. Feng, Y. Wu, B. Surigalatu, X. Zhang, and K. Chang, "Large transparent display based on liquid crystal technology," *Applied Optics*, vol. 59, no. 16, pp. 4915–4920, 2020 (cit. on p. 6).

[37] J.-b. Lee, I. Yanusik, Y. Choi, *et al.*, "Automotive augmented reality 3d head-up display based on light-field rendering with eye-tracking," *Optics Express*, vol. 28, no. 20, pp. 29 788–29 804, 2020 (cit. on p. 8).

[38] J.-P. Stauffert, F. Niebling, and M. E. Latoschik, "Latency and cybersickness: Impact, causes, and measures. a review," *Frontiers in Virtual Reality*, vol. 1, p. 582 204, 2020 (cit. on p. 35).

[39] R. Varshneya, R. Draper, J. Penczek, B. Pixton, N. F, and P. Boynton, "50-4: Standardizing fundamental criteria for near eye display optical measurements: Determining the eye-box," *SID Symposium Digest of Technical Papers*, vol. 51, pp. 742–745, Aug. 2020. DOI: 10.1002/sdtp.13975 (cit. on pp. 4, 7).

[40] Y. Hataji, H. Kuroshima, and K. Fujita, "Motion parallax via head movements modulates visuo-motor control in pigeons," *Journal of Experimental Biology*, vol. 224, no. 3, jeb236547, 2021 (cit. on p. 4).

[41] Y. Itoh, T. Langlotz, J. Sutton, and A. Plopski, "Towards indistinguishable augmented reality: A survey on optical see-through head-mounted displays," *ACM Comput. Surv.*, vol. 54, no. 6, Jul. 2021, ISSN: 0360-0300. DOI: 10.1145/3453157. [Online]. Available: https://doi.org/10.1145/3453157 (cit. on pp. 1, 7).

[42] T. D. Jalluri, G. M. Gouda, A. Dey, B. Rudraswamy, and K. Sriram, "Development and characterization of silicon dioxide clad silicon carbide optics for terrestrial and space applications," *Ceramics International*, vol. 48, no. 1, pp. 96–110, 2022 (cit. on p. 6).

[43] J. Gao, W. Ji, F. Chang, *et al.*, "A systematic survey of general sparse matrix-matrix multiplication," *ACM Comput. Surv.*, vol. 55, no. 12, Mar. 2023, ISSN: 0360-0300. DOI: 10.1145/3571157. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3571157 (cit. on p. 32).

[44] R. Liu and N. Choi, "A first look at wi-fi 6 in action: Throughput, latency, energy efficiency, and security," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 7, no. 1, Mar. 2023. DOI: 10.1145/3579451. [Online]. Available: https://doi.org/10.1145/3579451 (cit. on p. 34).

[45] M. N. Mustafa, M. A. A. M. Abdah, A. Numan, A. Moreno-Rangel, A. Radwan, and M. Khalid, "Smart window technology and its potential for net-zero buildings: A review," *Renewable and Sustainable Energy Reviews*, vol. 181, p. 113 355, 2023 (cit. on p. 6).

[46] *Perf(1) linux user's manual*, Linux Foundation, Jun. 2024 (cit. on p. 28).

[47] R. Romeiro, E. Eisemann, and R. Marroquim, "Retinal pre-filtering for light field displays," *Computers & Graphics*, vol. 123, p. 104 033, 2024, ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2024.104033. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849324001687 (cit. on p. 11).