

Supplementary Material for Versatile Framework for Low-Cost Parallax Multi-View 360° Displays

Petar Pjanic;
Independent Researcher; Switzerland;
petar.pjanic@gmail.com

Iva Salom;
Institute Mihajlo Pupin, University of Belgrade; Serbia;
iva.salom@pupin.rs

Igor Salom;
Institute of Physics Belgrade, University of Belgrade; Serbia;
isalom@ipb.ac.rs

Maja Dukic;
Independent Researcher; Switzerland;
maja.dukic.pjanic@gmail.com

Miomir Djukic;
Independent Researcher; Serbia;
dmiomir@gmail.com

This document contains supplementary materials for the paper "Versatile Framework for Low-Cost Parallax Multi-View 360° Displays." It includes a detailed explanation of the rendering workflow steps.

1 Rendering Workflow Steps

1.1 Strips formation

In this section, we explain how the strips depicted in the main manuscript in Figure 3. are formed. The number of strips on the screen directly correlates with the number of different views we aim to display.

Maeda et al. [1] demonstrated that the maximum

number of distinct views displayable is dependent on the screen's refresh rate and the display's rotation speed. The maximum number is calculated using the formula $n = \frac{rr}{s}$, where rr represents the refresh rate, s the rotation speed in revolutions per second (rps), and n the number of different images that can be displayed in one rotation.

For instance, suppose we wish to display only 4 different views. Since the image we observe is formed within a quarter of the full rotation of the screen, each individual strip will be as wide as the screen itself. In this scenario, we would just change the rendered view every quarter rotation. Now, let's consider a scenario where we aim to display 12 different views.

In this case, as the image formation still occurs within a quarter of a rotation and we cannot merely cycle through the render views as before. This is because merely cycling the content might present the observer with the wrong viewing direction. Instead, the screen will contain 3 strips, as depicted in Section 3. and Figure 3. in the main manuscript. The formula for determining the number of strips in this scenario is $s = n/4$.

Eq. 1 and 2 demonstrate how to calculate the angular start and the end position for a given strip, depending on the number of images we want to show. Then, the angular start and end positions are converted to screen positions using Eq. 3. and 2. in the main manuscript.

$$\alpha_{st}^i = \frac{\pi}{4} - \frac{4 \cdot (i-1)}{n} \cdot \frac{\pi}{2} \quad (1)$$

$$\alpha_{end}^i = \frac{\pi}{4} - \frac{4 \cdot (i)}{n} \cdot \frac{\pi}{2} \quad (2)$$

In this context, α_{st}^i represents angular start of the i strip, α_{end}^i represents angular end of the i strip, n is the number of different views that we aim to display and i denotes the index of the strip on the screen.

Figure 1 showcases two different strip configurations: one where the observer is positioned close to the rotating display, and another where the observer is farther away. In both instances, the screen contains 5 strips. Notably, when the observer is situated further from the rotating screen, the strips on the sides are significantly wider.

1.2 Strips offset

As demonstrated in Section 3. and Figure 3. in the main manuscript, as the display rotates, the strips will shift over time from left to right. This movement is crucial to ensure that each observer sees the correct image corresponding to their viewing direction.

The offset of the strips is determined by the rotation speed of the device and the refresh rate of the display. For instance, consider a scenario where we

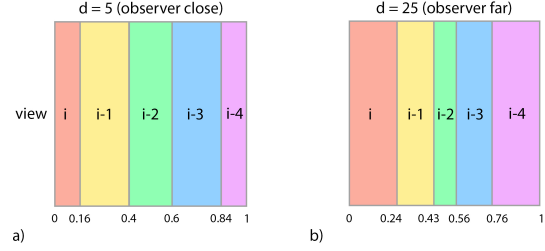


Figure 1: The strips corresponding to specific viewing angles depending on a distance of a observer to the display.

aim to display 12 different views using a 120 Hz display, with the display rotating at 10 revolutions per second. As outlined in Section 1.1, this setup results in the display containing 3 strips. In this case, as illustrated in Figure 3. in the main manuscript, with each update of the screen, a strip will jump to the position on its right, meaning strip i will move to the position of strip $i-1$.

Now, let's assume we again want to display 12 different views with the same 120 Hz display, but the display rotates only 9 times per second. In this scenario, merely moving the strip from position i to $i-1$ is not sufficient. Due to the rotation speed not being in sync with the refresh rate and the number of views we wish to display, the observed image will start to drift. For a specific viewing direction intended to show image i , this drift will cause the image to continuously shift, eventually displaying images for the views $i-1$, $i-2$, and so on. This drift will manifest as the observer object slowly rotating to the left. Similarly, with the same setup but a motor speed faster than 10 rotations per second, the drift will occur to the right. Likewise, if the refresh rate deviates from exactly 120 Hz, we will experience similar drifts.

In our setup, we do not have control over the refresh rate, and we are not employing Proportional-Integral-Derivative (PID) control for precise and consistent motor speeds as suggested in Jones et al. [2][3]. However, as illustrated in Figure 2, we can utilize a continuous offset, rather than a discrete one, to compensate for drift effects. For instance, in our example where the motor moves at 9 rotations per

second, instead of shifting from position i to $i-1$, we can offset position i to some position left of $i-1$. This adjustment allows us to counteract the drift, ensuring the image remains stationary.

Eq. 3 and 4 demonstrate how to calculate the offset for a given strip, taking into account the rotation speed, the refresh rate of the screen. Additionally, this offset can be manually adjusted, for instance, if the screen's refresh rate differs from the target value. Moreover, this approach can be used to correct for skipped frames. For example, if a frame is skipped, it would typically result in a sudden rotation of the object to the right. In such cases, the offset can be employed to correct this abrupt rotational jump.

$$n = \frac{rr}{s}. \quad (3)$$

$$of = (fr + dr + co) \bmod n. \quad (4)$$

$$\alpha_{st}^i = \frac{\pi}{4} - \frac{4 \cdot (i-1 - of)}{n} \cdot \frac{\pi}{2}. \quad (5)$$

$$\alpha_{end}^i = \frac{\pi}{4} - \frac{4 \cdot (i - of)}{n} \cdot \frac{\pi}{2}. \quad (6)$$

In these equations, n represents the number of different views we can display, s is the actual speed of the display, and rr refers to the actual refresh rate of the display. The variable i denotes the index of the strip on the screen, of is the calculated offset of the strip, and fr is the index of the current rendered frame. Additionally, dr signifies drift compensation, and the term co is used for compensation in cases of skipped frames.

Furthermore, we can use the timestamped external synchronization signal as depicted in the main manuscript to additionally stabilize the image and prevent drifts. As shown in the main manuscript, a Raspberry Pi is used to wirelessly send this timestamped external synchronization signal to the display (smartphone) when the rotating platform is in a specific rotational position. The offset calculation

is performed on the smartphone. Ideally, this signal should reset the offset (of) to a predetermined value ($predVal$). However, since the signal is sent wirelessly, significant random latency can occur. To account for this, the signal is timestamped and compared to the time it is received on the display (smartphone). When the timestamped signal is received, Eq. 7 illustrates how we use that signal to reset the offset:

$$of = predVal + 4 \cdot \frac{timedisplay - timesignal}{rotationPeriod}. \quad (7)$$

In this equation, of represents the calculated offset of the strip, $rotationPeriod$ is the time it takes to complete one full rotation of the display, $timedisplay$ is the current time on the display (smartphone), and $timesignal$ is the time when the signal was sent from the Raspberry Pi.

However, it's important to note that this approach has its limitations. For instance, if the speed of the motor is significantly faster than it should be, say five times faster, the compensation may not suffice. In such cases, the image could start to jitter.

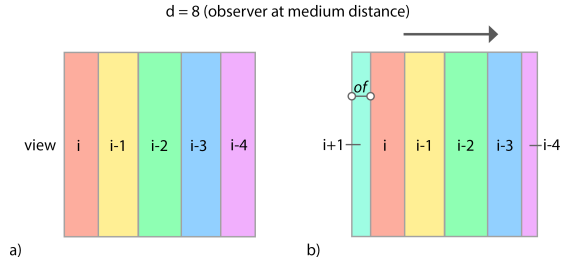


Figure 2: The schematic illustrates the offset corresponding to a specific rotational position of the cylinder.

1.3 Compensating for non-instantaneous screen-update

In most cases, the screen update process is not instantaneous, presenting a challenge to our method of dividing the screen with vertical stripes, as described in the previous section.

Consider a scenario where the screen updates from top to bottom. In our setup, the observer sees only one vertical strip on the display at any given time. However, due to the non-instantaneous nature of the refresh rate, the top of the vertical strip might display a new frame, while the bottom might still show the previous frame. As the cylinder rotates, the visibility of the previous frame diminishes until the entire strip is updated with the new frame. This phenomenon is depicted in Figure 3, where diagonal lines, rather than vertical lines, appear during the screen update. This effect can potentially undermine our approach, which is based on vertical strips.

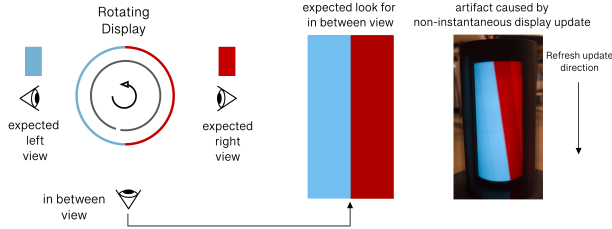


Figure 3: The influence of the refresh rate screen-update on the strip shape is demonstrated in this setup. For half of the rotation, a blue screen is displayed, and for the other half, a red screen is displayed. When viewing the display from the side, if the screen’s refresh is instantaneous, one would observe the left half blue and the right half red with a straight dividing line in the center. However, due to the screen updating from top to bottom, the observed dividing line appears diagonal instead.

Figure 4 demonstrates how to compensate for this effect. If the screen updates from top to bottom, we can adapt our approach by using diagonal strips instead of vertical ones. The slope of these diagonal strips will be determined by the duration it takes to update the screen.

Similarly, if the screen updates from left to right, a corresponding modification can be made to the strips. In this scenario, we would maintain vertical strips but adjust their positions accordingly.

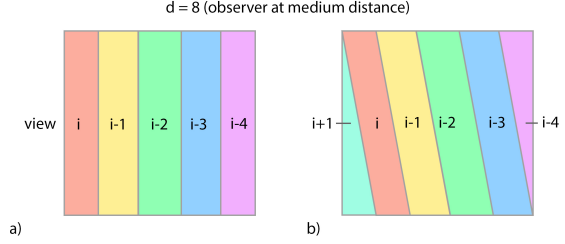


Figure 4: The schematic b) demonstrates how to adjust the strips depicted in a) to compensate for the non-instantaneous screen-update.

1.4 Perspective compensation

When observing lines on the screen through the slit, they are projected into the viewing space, leading to distortions. A primary issue is the ‘*barrel effect*’, which causes lines at the edges to appear smaller and compressed, as depicted in Figure 5.

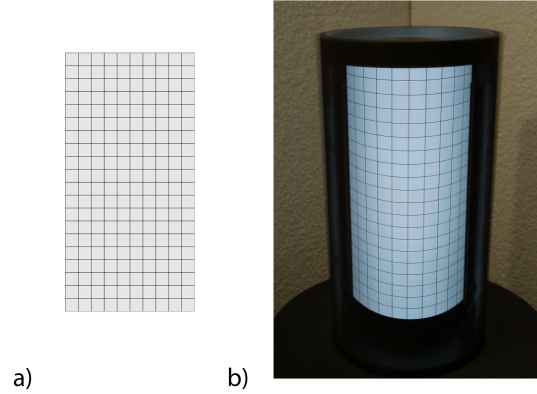


Figure 5: The demonstration of the *barrel effect* is shown, where a) is the uniform grid displayed on the screen, and b) is a photograph of that grid as seen with the cylindrical prototype. With the cylindrical prototype, it can be observed that the squares in the middle appear larger compared to the squares on the sides.

To counteract the *barrel effect*, we employ perspective compensation functions, derived from Eq. 3. in the main manuscript. These functions are initially

applied directly to the content we aim to observe. The inputs for this function are x , representing the line as seen in the viewing space, d , denoting the distance of the observer from the center of the cylinder, and r , which is the radius of the cylinder. The output, y , corresponds to the line on the rotating screen.

$$t = \arctan\left(\frac{rx}{d - r\sqrt{1 - \tan(\arcsin(x))^2}}\right) \quad (8)$$

$$y = \tan(\arcsin(x) + t)$$

Additionally, to correct for the effect of lines appearing smaller, we apply a second perspective compensation based on the line's depth. In this context, x represents the line in the viewing space, while y , calculated from Eq. 8, corresponds to the line on the rotating screen. The output k is the enlargement factor applied to the y line, taking into account the observer's distance d from the center of the cylinder and r , the cylinder's radius.

$$k = 1 + \frac{y \cdot x}{d \cdot r}. \quad (9)$$

1.5 In-between views and strip blending

The method we've outlined works best when the observer is positioned exactly where the refresh rate, rotation speed, and image strips align, allowing them to see the image meant for that specific viewing angle. In our setup, there are n such precise positions, where n is the total number of views displayed to the observer. Figure 6 demonstrates that in positions that are not exactly aligned, such as those between positions i and $i + 1$, the observer will see a mix of both views. Here, each strip displays parts of both the i th and $(i + 1)$ th views. The effect is somewhat like viewing between two images; we will refer to this effect as *half-viewing*. It is similar to being between two different images, where the closer we are to one view, the more dominant it appears.

If the screen has a very high refresh rate and can display many views (as detailed in Section 1.1), these

transitional strips are not very noticeable. However, with lower refresh rates and larger differences between views, the transition between strips becomes more evident, creating a noticeable dividing line.

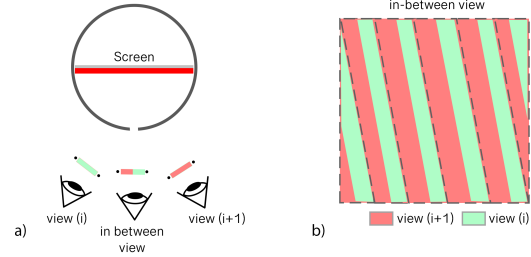


Figure 6: The figure illustrates an observer positioned between views i and $i + 1$ observing alternating strips corresponding to those views. If the observer is closer to one view, then the strips associated with that view would become visually larger.

This blending of views, a kind of 'half-viewing', is similar to an older image-based 3D rendering technique for creating smooth transitions between views. In our case, the blend between views changes based on the observer's position.

We compare two approaches for blending between strips:

- *Spatial Color-Blending*: This method blends the colors of adjacent strips, effectively reducing the visibility of hard lines. While this approach generally smooths the transition, some content may still exhibit noticeable discontinuities.
- *Angular Over-Blending*: Here, we create more strips than can actually be displayed. Instead of merely blending the colors of the strips, this method involves blending using rendered views that lie between the strips.

Figure 7 illustrates the differences between *spatial color-blending* and *angular over-blending*. *Spatial color-blending* is quick and efficient, while *spatial blending* offers a more continuous image that may be preferable to some observers. However, *angular over-blending* necessitates rendering additional

views, which could pose challenges for real-time applications. Notably, these two approaches can be combined, allowing for the rendering of slightly more views with spatial color-blending applied between them. We present the comparison between these two approaches in the main manuscript Figure 10.c.

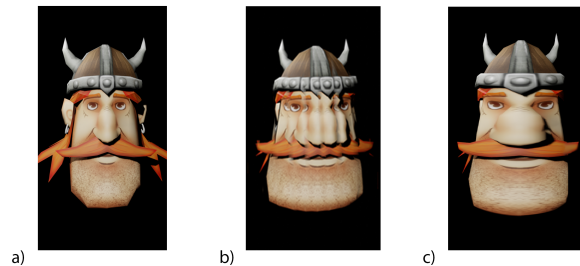


Figure 7: Two methodologies for strip transition and blending to produce the final displayed image are presented: a) an image of the Viking from the front presented for reference and comparison, b) a *spatial color-blending* approach constructed from 32 rendered views compiled into strips that are subsequently blended, and c) an image formulated from 512 rendered views, employing a *angular over-blending* approach to avoid repetitive structures.

References

- [1] H. Maeda, K. Hirose, J. Yamashita, K. Hirota, and M. Hirose, “All-around display for video avatar in real world,” in *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.*, pp. 288–289, 2003.
- [2] A. Jones, I. McDowall, H. Yamada, M. Bolas, and P. Debevec, “An interactive 360° light field display,” in *ACM SIGGRAPH 2007 Emerging Technologies*, SIGGRAPH ’07, (New York, NY, USA), p. 13–es, Association for Computing Machinery, 2007.
- [3] A. Jones, I. McDowall, H. Yamada, M. Bolas, and P. Debevec, “Rendering for an interactive 360° light field display,” *ACM Trans. Graph.*, vol. 26, p. 40–es, jul 2007.