Received: 12 June 2022 | Accepted: 20 June 2022

DOI: 10.1002/cav.2099

SPECIAL ISSUE PAPER

WILEY

# Dynparity: Dynamic disparity adjustment to avoid stereo window violations on stationary stereoscopic displays

Christoph Schröder-Dering | Gabriel Zachmann | René Weller

Computer Graphics and Virtual Reality, University of Bremen, Bremen, Germany

**Correspondence**
Christoph Schröder-Dering, Computer Graphics and Virtual Reality, University of Bremen, Bremen, Germany.
Email: schroeder.c@uni-bremen.de

**Abstract**
We propose a novel method to avoid stereo window violations at screen borders. These occur for objects in front of the zero parallax plane, which appear in front of the (physical) screen, and that are clipped for one eye while still being visible for the other eye. This contradicts other stereo cues, particularly disparity, potentially resulting in eye strain and simulator sickness. In interactive and dynamic virtual environments, where the user controls the camera, for example, via head tracking, it is impossible to avoid stereo window violations completely. We propose Dynparity, a novel rendering method to eliminate the conflict between clipping and negative disparity, by introducing a nonuniform stereoscopic projection. For each vertex in front of the zero parallax plane, we compute the stereoscopic projection such that the parallax approaches zero toward the edge of the screen. Our approach works entirely on the GPU in real-time and can be easily included in modern game engines. We conducted a user study comparing our method to the standard stereo projection on a large-screen stereo wall with head tracking. Our results show significantly reduced simulator sickness when using Dynparity compared to the standard stereo rendering.

**KEYWORDS**
stereo window violation, stereoscopic 3D, user studies, visual discomfort

## 1 | INTRODUCTION

Stationary stereoscopic displays such as big screens in cinemas, 3D TVs, or other stereoscopic projection walls offer several advantages over HMDs, for example, resolution, wearing comfort, and so forth. One of the problems of stationary displays, however, is the so-called stereo window violation, which occurs for objects that are to appear in front of the screen while, at the same time, are being clipped at the edge of the projection screen. Similarly, objects close to the border could be visible in the view frustum of one eye while getting clipped for the other eye, which results in binocular rivalry.[1]

In 3D movies, cinematographers take great care to keep objects with negative disparity close to the center of the screen.[2] Other automatic approaches simply remap the whole scene's depth or edit the content offline. In real-time environments, though, such as product reviews on a powerwall[1] with head tracking, the viewer's eye position is

[1]Powerwalls are large, stationary, stereoscopic displays that display content for one or more users, each of which must be tracked, with physically correct projections for the individual users.

not known in advance and therefore neither of these approaches works. While the scene content could be optimized to reduce stereo window violations, this severely limits or eliminates the space in front of the screen for object placement.

We present *Dynparity*, a novel algorithm to avoid stereo window violations at the view frustum's borders in off-axis stereo rendering. Dynparity is specially designed for interactive dynamic scenes, where the viewing position is not known in advance but also works for static setups. The main idea is to *dynamically* adjust the disparity on a per-vertex level. To do so, we modify the perspective projection such that the disparity of vertices in front of the zero parallax plane approaches 0 as they approach the left or right border of the view frustum, but it maintains the correct disparity in the center of the screen.

Throughout this work, we use the term *disparity* to describe the on-screen pixel parallax, following the convention by Terzić and Hansard.[3] In our case, it is directly related to the retinal disparity, as we use head-tracking to render the correct views.

Dynparity is real-time capable and easy to implement. It is very well suited for a single-pass vertex shader; hence it runs entirely on the GPU, with no further modification of the underlying application required. We have exemplary integrated it into a modern game engine, the Godot engine.

Moreover, we have investigated Dynparity in a user study with 31 participants. In this study, we focused on visual discomfort and user preference. Our results show that the adjustments made with Dynparity are hardly recognized by the users, while it simultaneously reduces symptoms of simulator sickness significantly compared to standard stereo rendering.

## 2 | RELATED WORK

Our main goal is to reduce the visual discomfort when viewing stereoscopic 3D displays by adjusting the disparity. Terzić and Hansard[3] give a good overview of sources of visual discomfort and possible ways to avoid or reduce it.

There is a lot of research that investigates disparity remapping for movies and television content. Methods in this field can be divided into two categories. First, some approaches globally scale or shift the disparity of a whole frame or scene. Often, these methods aim to move the important objects into the display plane to maximize the comfortable disparity range.

A simple way to avoid any adverse effect from negative disparity is to move the virtual scene completely behind the zero parallax plane and, hence, the display, as Xu et al. propose in their work.[4] In order to convert 2D video footage into 3D, Chen et al.[5] classify images as background or foreground dominant. For background dominant images, they shift the disparity so that the nearest part is on the focus plane. For foreground dominant images, they optimize the disparity histogram until it peaks at zero disparity. Similarly, for their stereo panoramas, Pritch et al.[6] describe an algorithm to narrow the baseline for closer scenes and use a larger baseline for faraway scenes.

Another approach is to remap the disparity values still for the whole scene but in a nonlinear way. Ide and Sikora[7] formulate a nonlinear disparity scaling that ensures correct depth perception when scaling video to different screen sizes. Xu et al.[8] reimagine contrast stretching for depth maps to increase the stereo acuity near the display plane.

Second, some methods warp parts of the scene individually. Lin et al.[9] use saliency maps to avoid distorting important image regions and further add cropping to avoid stereo window violations. Lang et al.[10] combine linear and nonlinear terms for global frame adjustments, as well as local features like the image and temporal gradients for application in videos.

The work mentioned before focuses on today's stereoscopic media, which is often not personalized but viewed together, for example, in cinemas. Other methods cater to individual viewers and therefore can support real-time, interactive experiences: Ware et al.[11] allowed the user to scale the disparity and noticed that the user's preference depended on the scene content. They further proposed an algorithm that automated the adjustment. Sun and Holliman[12] use the Z-buffer to scale the depth range dynamically. They show that filling the comfortable disparity range by varying the inter-camera distance can be beneficial over a fixed baseline. Didyk et al.[13] propose a model that predicts the effect of disparity and could be used to adjust disparities based on the individual subject's stereoacuity.

OSCAM by Oskam et al.[14] adjusts the stereo projection parameters dynamically based on the depth map in real-time. Furthermore, to avoid sudden changes in appearance, they propose a special interpolation of the parameters. Celikcan et al.,[15] and Koulieris et al.[16] both propose systems where they move the zero parallax plane toward the objects the user most probably looks at. They determine these objects by a scoring and machine learning algorithm, respectively. Both methods require preprocessing: The scoring needs to be determined per object; for the machine learning, they train an attention model for a specific scene and require eye-tracking during this step. Recently, Avan et al.[17] showed that manually authored mappings of stereo parameters based on the camera location in the virtual environment could improve the perceived depth and picture quality while maintaining the visual comport in HMDs. For their method, producers have to manually annotate every part of the virtual world.

Methods that do not modify the camera parameters, such as the work from Lou et al.[18,19] deform the scene geometry to reduce motion sickness in VR. Their method does not directly address stereo window violations, though.

As our method, like others, modifies the disparity, it is important to understand that this could influence distance estimation. Bruder et al.[20] find the distance between the user and the screen and whether the disparity is positive or negative to be the main factors for distance estimation. Further, it is well known that stereopsis is best in the center of the user's visual field and declines significantly toward the edges.[21-23] On large projection screens, such as the Powerwall we use in our experiment, the visual field is smaller than the screen. Therefore, we assume that the part of the screen the user focuses on most is in the center of the screen. In contrast to other methods, Dynparity keeps the correct disparity in this central region.

To the best of our knowledge, no prior work makes a per-vertex disparity adjustment in real-time, interactive applications to avoid stereo window violations at screen borders.

## 3 | OUR APPROACH

Physically correct real-time rendering for powerwalls requires asymmetric projection matrices for each eye that depend on the user's head position and orientation at each frame.[24] Typical disparity adjustments either involve changing the inter-pupillary distance (IPD) or changing the distance between the eye and the projection plane to redistribute the negative and positive disparity, which both change the disparity for the whole scene.

The idea of our approach is to change the disparity of objects only *locally*, that is, we aim at preserving the pop-out effect of things close to the center of the image while minimizing the stereo window violation close to the borders of the screen. Consequently, we need a *local* approach that depends on the composition and the viewpoint of the scene.

One option to achieve such an effect is to warp the left and right eye images after rendering. Warping can be applied locally and therefore does not necessarily change the whole scene. For example, such a typical post-processing warping approach could be to save a mask with parts that are in front of the projection plane and store their disparity. However, warping elements in the foreground as a post-processing step results in holes in the background that need to be filled by warping neighboring pixels, which would cause distortions in the background.

The idea of our Dynparity algorithm is to avoid such a post-processing image warping, but to adjust the scene *before* the rasterization of the image, that is, we perform it during the vertex processing in the *vertex shader*. We will detail this idea in the next section.

### 3.1 | Basic idea

In the previous section we already mentioned that we do all our computations directly in the *vertex shader*. The main idea is to keep the disparity unchanged for vertices close to the center of the screen and reduce the disparity the closer a vertex is to the border (Figure 1). When a vertex is projected directly on the edge of the view frustum, its disparity should be zero. Obviously, we only have to adjust vertices that are *in front* of the screen, that is, for vertices with a negative disparity, because only those vertices are affected by clipping stereo window violations.

So for each vertex, we have to decide whether it has positive or negative disparity, and in the latter case, we have to adjust its position. And this is exactly what our shader does.

We start by rendering the scene with the conventional asymmetric stereo projection. In the vertex shader though, we shift both eyes to the cyclops-eye position first. We do the shifting in the vertex shader to keep the eye positions correct for the CPU based frustum culling and the calculations for shadow-mapping. Then, we first determine whether a vertex $\vec{v}_i$ is in front or behind the screen:

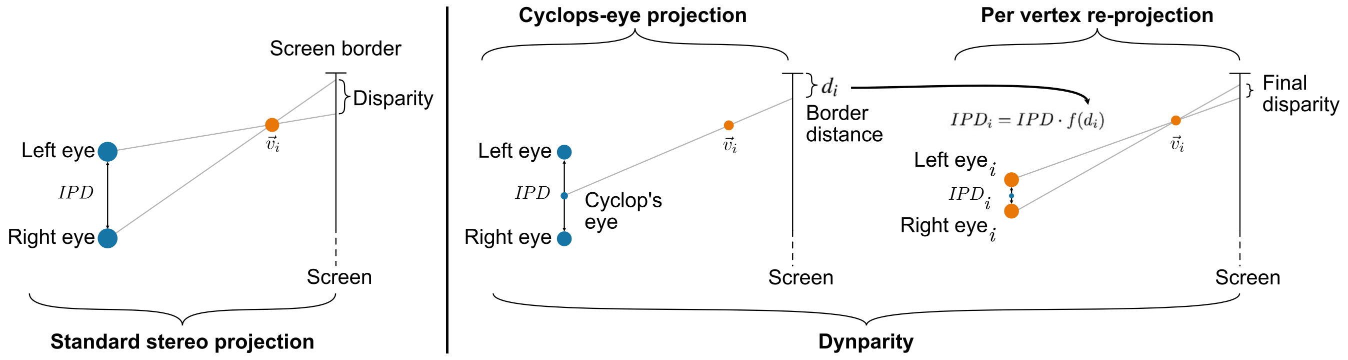

FIGURE 1 The left part shows standard off-axis stereo projection of one vertex $\vec{v}_i$. Dynparity extends the rendering with a per-vertex reprojection as shown in the right part. We adjust the IPD for each vertex $\vec{v}_i$ relative to the distance $d_i$ to the left or right border of the view frustum, which is determined from the cyclops-eye projection first. Vertices in the center of the view frustum have a large distance $d_i$ and as such are rendered with normal IPD.

$$\vec{n} = \frac{(\vec{s}_b - \vec{s}_a) \times (\vec{s}_c - \vec{s}_a)}{\|(\vec{s}_b - \vec{s}_a) \times (\vec{s}_c - \vec{s}_a)\|}, \tag{1}$$

$$z = \vec{n} \cdot (\vec{v}_i - \vec{s}_a), \tag{2}$$

where $\vec{s}_a, \vec{s}_b, \vec{s}_c$ are lower left, lower right and upper left corners of the screen in world-space, and $\vec{v}_i$ is the vertex in world-space. In a right-handed coordinate system like in Open-GL, $z$ will be positive for vertices in front of the screen. Of course, one could also do it in other coordinate systems, for example, the normalized device coordinates (NDC). Vertices behind the screen are transformed normally with the traditional stereo projection matrix.

For all vertices $\vec{v}_i$ in front of the display, that is, that would result in negative disparity, that we transformed into their particular NDC $\vec{q}^{\,i}$, we compute their distance $d_i$ to the edges of the view frustum. The results of our prestudy suggested that the most visual discomfort is caused by the asynchronous clipping of the left and right eye images at the horizontal edges of the screen. In order to limit the amount of added distortion, we therefore only consider the distance to the left and right edges of the view frustum.

$$d_i = 1 - |q_x^i|. \tag{3}$$

Our goal is to modify their positions so that directly at the edges, these vertices have no disparity, that is, they appear to be located on the zero parallax plane. In contrast, vertices close to the center of the screen, that is, $d_i \approx 1$, should retain their negative disparity.

Technically, the output of the vertex shader is the position of the current vertex in the clip-space. Hence, an easy way to compute the modified positions is to change the IPD in the projection matrix. So, for $\vec{v}_i$ with negative disparity, we set:

$$\text{IPD}_i = \text{IPD} \cdot f(d_i). \tag{4}$$

The transfer function $f(d_i)$ could be any interpolating function with $f(1) = 1$ and $f(0) = 0$.

In the final step, we project the vertex again, using the new projection matrix for the respective eye that is positioned IPD$_i$ units away from the other eye. In our shader, we compute the modified vertex positions with the new projection matrix. Thus, all computations are done in a single pass.

In a prestudy, we compared three transfer functions for Equation (4) which we show in Figure 2. While the linear mapping produces the least amount of high frequency visual distortion as the change in adjustment is constant over the entire view frustum, it also reduces the overall disparity the most. On the other hand, the sigmoid-like function preserves the most disparity overall but suffers from a large area of rapid adjustment, leading to more noticeable distortions. Lastly, we investigated the square root mapping. It produces fewer distortions in the center of the screen than the linear mapping while maintaining a slow adjustment rate until the edges of the view frustum. Consequently, we decided to use $f(d_i) = \sqrt{d_i}$ in our user study.

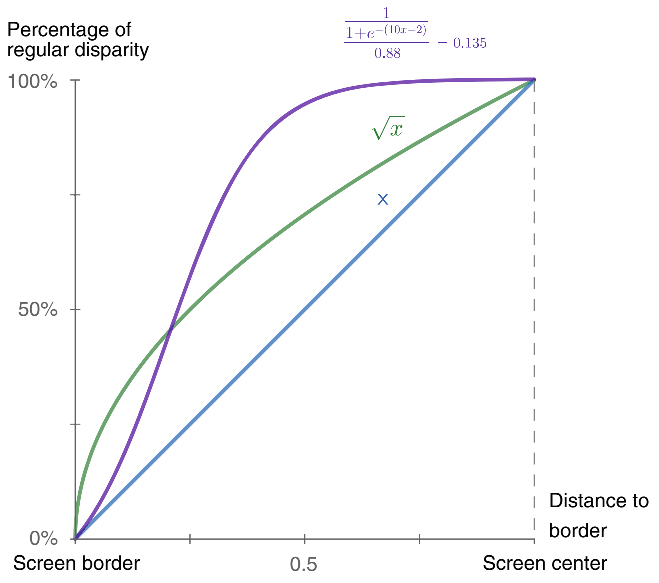

FIGURE 2 We considered three transfer functions to adjust the disparity based on the distance to the left or right border of the view frustum. The linear mapping shown in blue reduces the disparity the most. In our experiment, we chose the square root mapping depicted in green. It preserves more disparity overall with a small area of rapid adjustment. The sigmoid-like function in purple preserves more disparity in the center of the view frustum but suffers from a large area of rapid adjustment, leading to more noticeable distortions.

Figure 3, on the left, shows that the standard stereo projection: Objects close to the viewer, such as in the upper right, have a large disparity. The center image visualizes the transfer function: Vertices in the view frustum's center or behind the screen keep their disparity (dark blue/black), while those approaching the left and right borders are more and more adjusted (dark red). On the right, we see the resulting image after the reprojection: The disparity at the border approaches zero for objects in front of the zero parallax plane, as can be best seen in the upper right corner.

## 3.2 | Integration into Godot

We have implemented our Dynparity algorithm in a modern game engine, namely the Godot engine. We will briefly explain the actual implementation details which could also be useful for integration into other game engines.

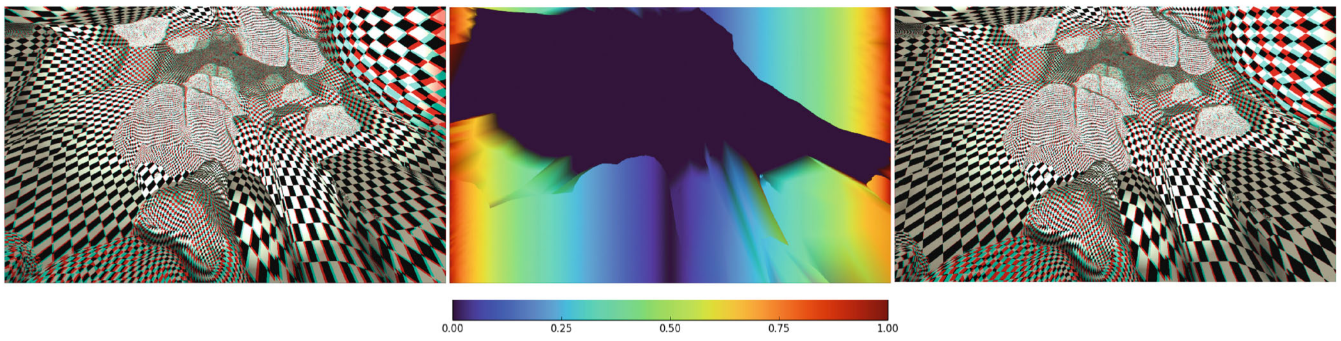

FIGURE 3 Our method, suitable for any projection-based or other stationary stereoscopic display, starts with a standard stereo rendering of the scene (left). Here we show the tunnel scene from our user study, with the boulder in the top right corner appearing in front of the zero parallax plane. We superimposed a checkerboard texture for this image to show the disparities better. In the vertex shader, we determine the distance to the left and right view frustum sides for vertices in front of the zero parallax plane, areas in black are behind the plane (center). Based on this distance, we reproject each vertex in the same rendering pass to reduce its parallax closer to the edges of the screen, which yields the final image (right). (The images can be viewed using red-cyan anaglyphic glasses to see the effect stereoscopically.)

Our integration into Godot consists of two parts: first, the implementation of an appropriate off-axis stereo projection supporting head tracking because this was not part of the original Godot code, and second, the implementation of our actual algorithm into the Godot shader structure.

We start with the description of our general VR plugin for conventional stereo projections, with the focus on head-tracked virtual environments on large projection walls. The plugin implements engine callbacks to configure the rendering pipeline and in each frame provides the position and projection matrices for each eye. Further, it communicates with the tracking system using VRPN to calculate the eye's positions. In addition to these standard VR plugin callbacks, our plugin also sets the new shader parameters for the Dynparity algorithm as described below.

In the second step, we modified Godot's main scene rendering logic to add the necessary parameters to the main uniform buffer that allows us to reconstruct the projection matrix in the vertex shader. More precisely, we added three corner positions of the display in world space, the cyclops-eye position, the current eye's offset vector, and a flag to enable and disable the reprojection. The eye offset contains both the IPD and head rotation.

Godot uses a single base scene shader with a placeholder where the user's custom shader code can be included. All materials in the engine use a dynamically derived version of this shader. Therefore, our modification is compatible with all engine effects that are compatible with standard stereo rendering. The core part of the implementation is given in Listing 1.

## 4 | USER STUDY

To evaluate the effect of our reprojection on users' perception and comfort as well as preference, we conducted a user study.

### 4.1 | Research questions and hypotheses

In our user study, we compare the adjusted rendering using Dynparity with the standard stereoscopic rendering that produces the known stereo window violations close to the edges of the view frustum. We decided to compare our method to standard stereo rendering only. First of all, it is recommended to keep the total time of the study below 40 min[25] to reduce fatigue effects, which would limit the number of methods we can compare. Second, as stated in Section 2, to our knowledge, no method explicitly deals with stereo window violations and does not change the physically correct projection in the center of the screen. Third, as standard stereo rendering is the most commonly used method on today's Powerwalls yet, and suffers from stereo window violations, we felt we had to include it in the comparison. In our user study, we want to answer the following research questions.

*RQ1: Do the participants prefer the Dynparity rendering over the standard stereo rendering with stereo window violations?*

In order to answer this research question, we have formulated two hypotheses. First, we assume that the effect of the Dynparity on the experience is not recognizable as long as the user is not informed about it. Thus, we formulate the first hypothesis as:

*H1: The participants do not recognize a difference when experiencing the virtual environment with Dynparity or standard stereo rendering, respectively.*

Notwithstanding H1, even if the users do not directly recognize a difference, we assume that the users will prefer the Dynparity rendering due to the reduced stereo window violations:

*H2: The participants prefer the Dynparity rendering.*

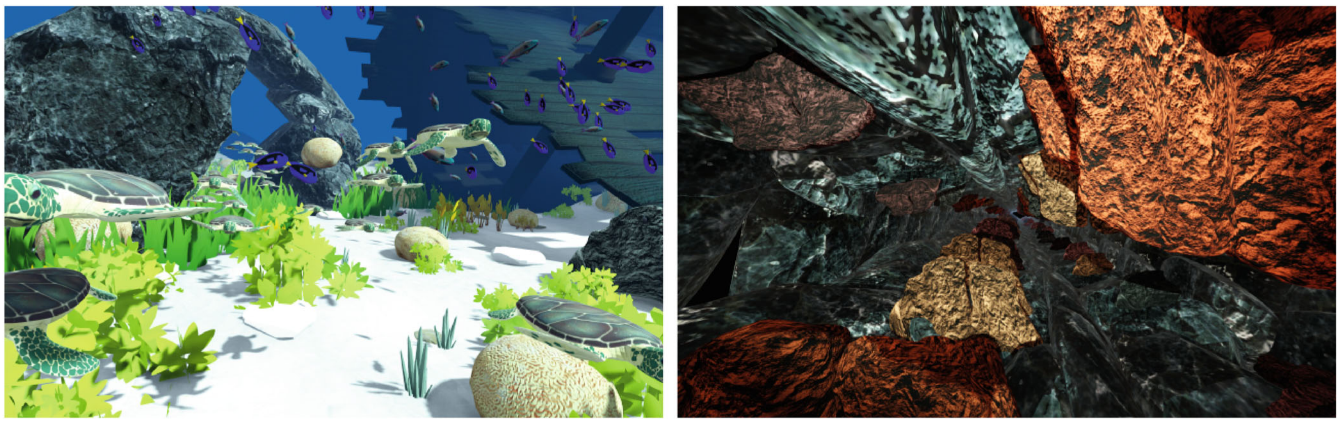

FIGURE 4 For the first part of the study, we used an underwater ocean scene with a rock arch and shipwreck in the background. Schools of fish swim from the background to the foreground but do not pass through the user's head position. Lines of turtles swim from the background toward the users and pass them to their left and right. In the second part, the user is transported very slowly through a rocky tunnel. The inner diameter is slightly smaller than the screen of the powerwall to ensure negative disparity on all sides. Thus, rocks frequently induce stereo window violations at the borders of the view frustum. Excerpts from both scenes are available in the supplementary video.

Our second research question is about the simulator sickness that stereo window violations can induce. Again, the Dynparity rendering should reduce this since it avoids stereo window violations at the edges:

*RQ2: Does Dynparity rendering reduce simulator sickness compared to standard stereo rendering?*

To answer this research question, we have formulated the corresponding hypothesis as:

*H3: The participants suffer from less severe simulator sickness symptoms when Dynparity rendering is enabled compared to standard stereo rendering.*

## 4.2 | Stimuli

For this study, we use two scenes as shown in Figure 4 and the supplementary video[2]: The first scene shows an underwater ocean environment with rocks, plants, and a shipwreck in the background. Schools of fish move freely in the scene but avoid swimming directly through the user to avoid discomfort due to very high negative disparity. To ensure the occurrence of retinal rivalry with negative disparity, a stream of turtles swims from the back of the scene toward the user and passes left and right of him.

In the second scene, the user is moved through a rocky, linear tunnel at a constant speed. Smaller and larger rocks with varying colors stick from the sides to the center of the tunnel. These rocks ensure repeated stereo window violations due to clipping at all four screen borders. We scattered gold coins throughout the tunnel to make the tunnel more exciting and reduce the fatigue from the repeated appearance. The initial camera position is horizontally centered in the tunnel and set to 3/5 of the tunnel's height. This placement ensures that no clipping and excessive negative disparity occurs throughout the experiment.

## 4.3 | Measurements

In order to measure the preference of the users in the ocean scene, we let them manually switch between the two conditions until they found their preferred configuration. We repeated this three times. We additionally asked for the preferred condition in the tunnel scene.

[2]https://cgvr.cs.uni-bremen.de/research/dynparity/

To measure the simulator sickness, we used the Simulator Sickness Questionnaire (SSQ).[26] We chose the SSQ with the two-factor evaluation proposed by Bouchard et al.[27] Contrary to the original three-factor evaluation, the two factors *Nausea* and *Oculomotor* have no cross-loading items with loadings below 0.4 and were extracted based on a user study with 371 participants. Furthermore, the study by Bouchard et al. is not limited to fighter pilots as subjects, which is a common critique of the original SSQ.[28] We used a five-point Likert scale in the questionnaire with the lowest, middle, and highest options labeled as no perception, light perception, and severe perception following recommendations from Blimberg et al. and Krosnick et al.[29,30]

Moreover, we only wanted to include participants that are able to see stereoscopic content well. Therefore, to measure the stereoscopic acuity, we used the Titmus Fly circle pattern test. We required a minimum depth discrimination of 100 arcs, following the recommendation for driving buses with passenger transport in Germany.[31] This level of stereopsis corresponds to the fifth out of the nine test squares in the commonly used Titmus stereo acuity test.

## 4.4 | Sample

We used a within-subject design where all subjects experienced standard stereo rendering and Dynparity rendering. We chose this over a between-subjects design as simulator sickness can differ vastly between different subjects, which would have required a much larger sample size. Furthermore, in order to minimize order and learning effects, we randomized all conditions accordingly.

Over 3 weeks, 36 volunteer participants completed the study procedure that took about 35 min. After the recording, we assessed the data quality. One subject was excluded due to a failure of the tracking system, and four subjects failed to meet the minimal stereoscopic discrimination of 100 s of arc in the circle test that we decided on before the study started.

According to the Titmus Fly circle test, all 31 remaining subjects (6 female, 25 male) included in the evaluation had normal or corrected to normal vision.

## 4.5 | Apparatus

The study was performed on a $4 \times 2.5\,\text{m}^2$ powerwall with rear projection at $2560 \times 1600$ @ 60 Hz resolution per eye, and wireless shutter glasses. For tracking the user's head position and rotation, an optical tracking system by Optitrack was used. The tracking system reported position and orientation at 120 Hz via gigabit ethernet using the VRPN[32] protocol. The position was filtered and projected ahead half a frame with a Kalman filter to prevent slight jitter and reduce the tracking latency. The filter parameters were estimated with a typical, prerecorded trajectory using the `find_optimal_momentum_filter` method from Dlib.[33]

Subjects sat on a chair that was horizontally centered and two meters in front of the powerwall. They were allowed to move their head freely but had to keep sitting on the chair to make all subjects see the same content and avoid clipping due to moving out of bounds. While scenes were shown on the powerwall, the room was completely darkened, and the light was only turned on to answer the questionnaires. This prevented reflections in the glasses and maximized the contrast on the powerwall.

## 4.6 | Protocol

In Figure 5, we provide an overview of the study procedure. In the following, we describe the different steps in more detail. After providing informed consent, the participants' IPD was measured, and their stereoscopic acuity was determined using the Titmus Fly circle pattern test. Next, we set the stereo projection to the individually measured IPD.

### 4.6.1 | Preference in dynamic scene

Then, the participants performed the first part of the study. The user watched the ocean scene, which was randomly started with standard or Dynparity rendering. With a single keypress on a computer keyboard, the subjects could switch between the rendering configurations, but they did not know what exactly changed. We asked the subjects to choose which of two configurations they preferred. Upfront, we informed them that there was no limitation on how often they switched and

| Preparation | RQ1: Preference | RQ2: Simulator Sickness |
|---|---|---|
| Introduction | Scene: Underwater | Scene: Tunnel |
| Informed consent | 3x 2AFC | 5 min watching w/o task, randomized Standard/Dynparity |
| IPD measurement | Short break | SSQ #2 |
| Titmus circle test | SSQ #1 → Baseline | 5 min watching w/o task, complementary condition |
| | | SSQ #3 + users explain their preference choice |
| 10 min | 5 min | 15 min |

FIGURE 5 Our study procedure comprises three blocks: First, the preparation is done. Then, we show the participants the ocean scene on the powerwall and ask them to choose their preferred configuration. They have to choose thrice, but we do not tell them that the differences are the same each time. At the end of the second block, we record the baseline SSQ. In the last block, we show them the tunnel scene and record the SSQ again. The participants all see the tunnel with standard and Dynparity rendering. Finally, we ask them about their preferred version of the tunnel and their reasoning and perceived differences.

how long they took. Further, we made clear that the only optimization criterion is their personal preference. When the subject pressed the configuration switching key, the screen faded to black for half a second and back to the scene. During the fading, we froze animations and movements in the scene. This fading hid flickering in the image when the rendering changed, which would otherwise make the nature of the change easy to spot.

To discriminate between strong preference and chance, we repeated this process three times. We told the subjects that we would calibrate three different parameters based on their choices to give the illusion that they would choose between six configurations. The intention was to make the subjects re-evaluate their preference each time instead of trying to repeat their previous choice.

### 4.6.2 | Longer exposure

The second part of the study investigated how long exposure to the two rendering modes influences simulator sickness. All subjects first answered the SSQ[26] as a baseline for the long exposure. The first experiment has variable duration; therefore, its influence on the SSQ is highly user-specific. So we conducted the baseline questionnaire only at this point in the experiment instead of at the beginning. With the baseline after the short break, we assume that the ocean scene has the same influence on all user's individual scores in the subsequent iterations of the questionnaire.

After the questionnaire, the subjects watched the tunnel scene, where they were instructed just to watch the scene until it automatically ended. The scene ended after 5 min, but we did not inform the user about the duration. After watching the scene, the subject answered the SSQ a second time. Finally, the same scene was shown again, followed by the last SSQ. The scenes were once rendered with standard stereo projection and the other time with Dynparity. The order of the condition was randomized.

After the last SSQ was answered, we asked the participants if they preferred the first or the second tunnel scene.

### 4.6.3 | Perception of difference

After the experiment, we further asked the participants to give a reason for their preference. We also asked them to describe any differences between the two configurations in the tunnel or the ocean scene they might have observed. We did not inform them about this question in advance or formulate it as a task to not pressure them to find differences at all costs, and secondly, to not influence the SSQ responses.

## 5 | RESULTS

For the evaluation, we used our unoptimized version of the Dynparity algorithm. The scenes used in the experiment both ran at the projectors' refresh rate of 60 Hz. Without vertical synchronization, the average frame time in the ocean scene was 15.03 ms ($\sigma = 0.10$ ms) for standard stereo rendering and 15.18 ms ($\sigma = 0.10$ ms) with Dynparity. In the tunnel scene the same time where 7.74 ms ($\sigma = 0.15$ ms) and 7.84 ms ($\sigma = 0.10$ ms), respectively. Dynparity recalculates the position of the left and right eye and their projection matrices based on the vertex's screen position; therefore, effects that depend on these variables must be recalculated. In Godot's standard material, only the user-programmed effects and the vertex's normal, tangent, and binormal vectors must be calculated twice. Therefore, the theoretical increase in run-time complexity is linear in the number of vertices visible in the view frustum. Furthermore, the increase per vertex is constant and is dominated by the cost of the custom shader code as the normal vector calculation is neglectable.

In the following, evaluate the results of our user study with respect to the two different research questions.

## 5.1 | Perception of difference in rendering

From the question about any differences between the configuration (see Section 4.6.3), we got a wide variety of answers: Most participants (20) did not notice a difference at all and said that they chose a random mode. Two found that one tunnel runs slower than in the other run, but there was no correlation to the rendering mode or if it was the first tunnel presented or the second. Two users reported that they chose the first tunnel because it was more boring the second time.

Three subjects reported that the standard rendering had a stronger 3D effect, but only two preferred that mode. Two participants noticed the distortions at the edge in the tunnel with Dynparity and preferred standard rendering. Another two subjects preferred Dynparity as it was more comfortable and less dizzying for them.

## 5.2 | Simulator sickness

Following the protocol in Section 4.6, we use the two-factor version of the SSQ proposed by Bouchard et al.[27]

The two factors are not normally distributed in our study. Since we compare three unmatched groups (baseline, standard rendering, and Dynparity rendering), we use the Quade test.[34] This is a nonparametric test that has higher power for less than five treatments compared to the alternative Friedmann test.[35]

Simulator sickness symptoms normally become more severe with longer exposure to VR content. Therefore, we first analyze the simulator sickness symptoms without considering the rendering mode (see Figure 6). When we compare the symptoms of the three questionnaires in the order they were filled, the Quade test reveals a significant difference $F(2, 60) = 3.499, p = .037$ in the nausea factor. A Quade post-hoc test using Holm/Bonferroni correction shows a significant difference between the baseline questionnaire and the last questionnaire the subjects answered ($p = .031$).

Next, we compare the SSQ answers after the different conditions. For the nausea factor (Figure 7), the Quade test reveals a significant difference $F(2, 60) = 3.431, p = .039$. A Quade post-hoc test using Holm/Bonferroni correction shows a significant difference between standard stereoscopic rendering and the baseline ($p = .03$). The baseline has the lowest nausea symptoms with a median of 0.61, whereas standard rendering has the highest median value of 1.83.

For the oculomotor factor (Figure 8), the Quade test reveals a significant difference $F(2, 60) = 3.847, p = 0.027$. A Quade post-hoc test using Holm–Bonferroni correction shows the significant difference between standard rendering and the baseline ($p = 0.049$) and between the standard rendering and Dynparity ($p = 0.049$). The oculomotor symptoms for Dynparity are the lowest with a median of 1.22, followed by the baseline (1.55), and are the highest for the standard rendering (2.08).

## 5.3 | User preference

In this section, we present our results concerning the users' preferences. As described above, we asked the users for their preference for both the ocean and the tunnel scenes.

During the first part of the study, the participants selected the preferred rendering mode without knowing what changed between the two options. Every participant had to choose three times. Regarding the sum of all choices, the users selected standard rendering 49 and Dynparity 44 times. The binomial test showed no significant difference.

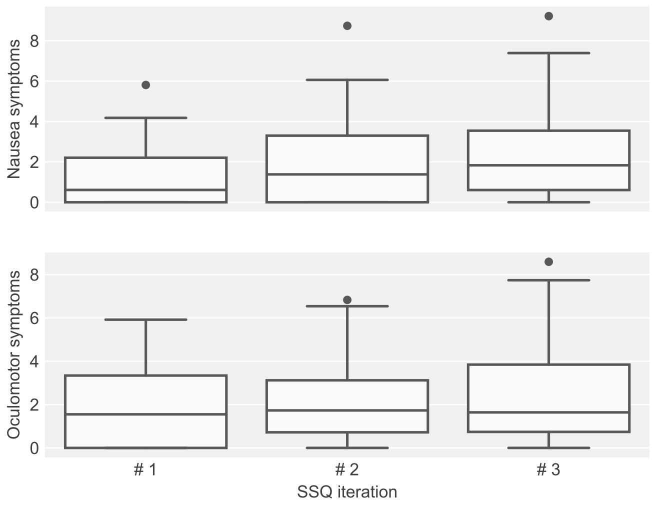

**FIGURE 6** Simulator sickness symptoms of all participants reported after the first (#1, baseline), second (#2), and third (#3) 3D stereoscopic experience, using the SSQ questionnaire. A higher score means worse symptoms. The median of the nausea symptoms increases with the duration of the experiment, while the oculomotor symptoms do not change much.

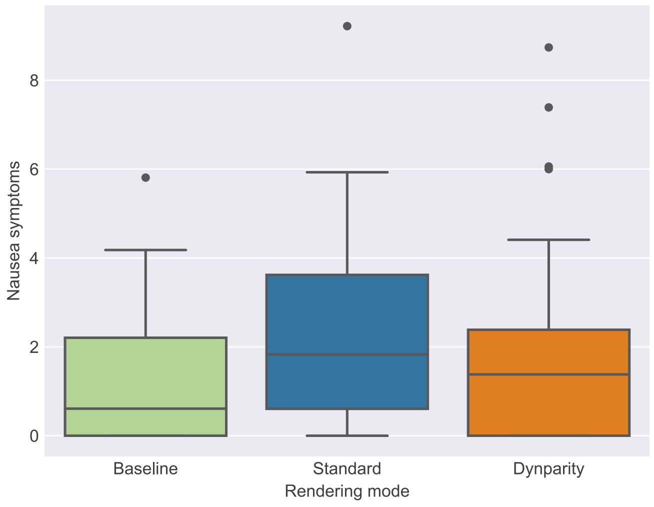

**FIGURE 7** Nausea symptoms of the baseline questionnaire before the tunnel scene and after the respective rendering mode in the same scene. A higher score means worse symptoms. Calculated after Bouchard et al.[27]

Moreover, if we additionally consider the individual choices per user, we see that the three choices were, in most cases, not coherent. The left plot in Figure 9 shows that only four subjects selected the standard rendering in all three choices. Similarly, only five users preferred our Dynparity rendering each time. Most users (15) chose the standard rendering in 2/3 times, and 7 participants chose Dynparity in 2/3 cases. When we split the users into two preference groups: those who selected the standard rendering in at least two out of three cases and those who selected Dynparity more often, the binomial test shows no significant difference between the selection frequency.

Additionally, we asked the users for their preference in the tunnel scene. As our study uses a within-subject design, there is always the possibility that subjects might get bored when watching the same scene twice. Our tunnel experiment could not observe a strong tendency toward the first or second run, as 16 subjects preferred the first run and 15 preferred the second run. Overall, 14 users preferred standard rendering, and 17 selected Dynparity (see Figure 9). In both cases, the binomial test showed no significant difference.

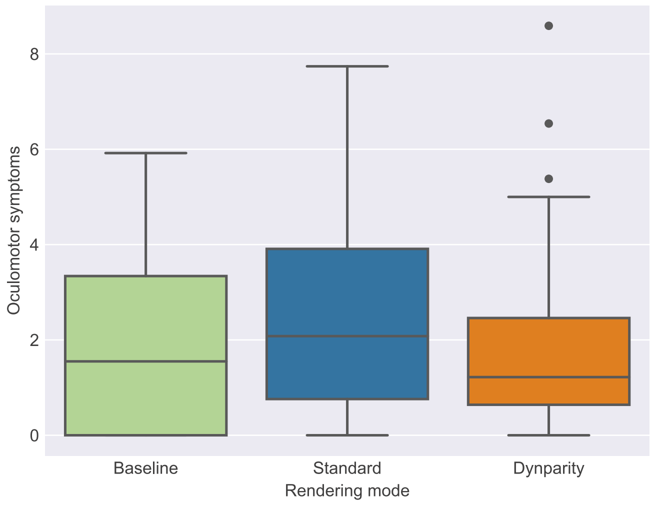

**FIGURE 8** Oculomotor symptoms after the baseline stereoscopic experience (green), and after the "tunnel scene," which would show frequent stereo window violations, both with standard stereoscopic projection (blue) and our Dynparity (orange). A higher score means worse symptoms. Calculated according to Bouchard et al.[27]

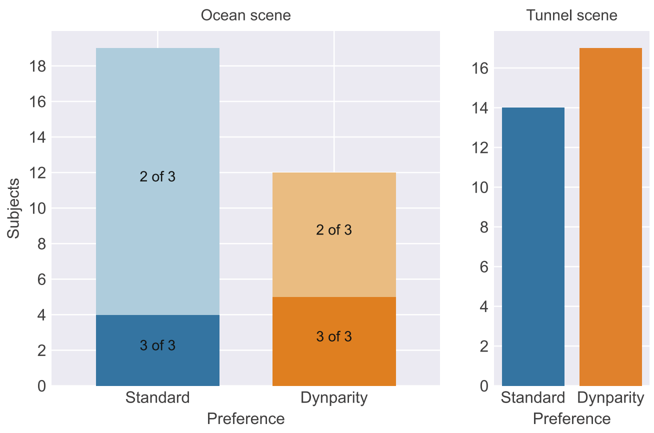

**FIGURE 9** Preferences of the different stereo projection methods for the ocean scene (left) and the tunnel scene (right), respectively. In the tunnel scene, more users preferred Dynparity over the standard stereo projection method. In the ocean scene, where the stereo violation occurs less frequently, users chose the standard stereo projection more often than Dynparity. Users had to choose three times. Only nine users had a clear preference (the "3 of 3" portions) for either projection method. Of the remaining (the "2 of 3" portions), 15 users showed a tendency toward the standard projection, while 7 preferred Dynparity.

## 6 | DISCUSSION

The results revealed significant differences in some cases. We will discuss them with respect to our research questions from Section 4.1.

### 6.1 | Perception of difference in rendering

In total, only 5 out of 31 participants noticed a difference between standard and Dynparity rendering. The perceived differences were a reduced 3D effect (3×) and artifacts at the borders of the screen (2×). In our scenario where users do not specifically look for differences, this supports our hypothesis H1.

## 6.2 | User preference

From the study data, we cannot deduce a clear user preference for standard or Dynparity rendering. While in the first part of the study, where the subjects could switch between the rendering modes, standard rendering was preferred more often (49×) than Dynparity (44×). However, there was no significant difference.

Similarly, only three more subjects preferred Dynparity over the standard rendering in the second part of the study. Hence, hypothesis H2 cannot be supported.

While some parts are always in front of the screen in the tunnel scene and produce negative disparity, only the turtles and fish are in front of the screen in the ocean scene. There could be a stronger preference for standard rendering in scenes where only a few objects move in front of the screen with a larger sample size. This would be in line with the recommendation given by Mendiburu[2] that small or fast-moving objects produce less discomfort than bigger and slower ones in case of stereo window violations. For scenes that have many stereo window violations, Dynparity could be preferred. However, in general, our research question RQ1 remains open.

## 6.3 | Simulator sickness

In the second part of our study, we measured the severity of nausea and oculomotor symptoms. Our baseline questionnaire captures the severity after the first part of the study. Therefore, it is no surprise that both factors do not start entirely near zero. As expected, the severity of the median of the symptoms rose over time, as we can see in Figure 6. Surprisingly, there was a slight decrease in oculomotor symptoms after watching the tunnel scene rendered with Dynparity.

For both factors, the standard rendering caused the most severe symptoms. Compared to the baseline, both nausea and oculomotor factors differed significantly. Between the standard rendering and Dynparity, only the oculomotor factor had a significant difference. The contributing items: fatigue, headache, eyestrain, difficulty focusing, difficulty concentrating, fullness of head, blurred vision suggest that Dynparity might reduce the load on the human visual system. Overall, Dynparity reduced the amount of simulator sickness. This means we can accept hypothesis H3.

In Figure 7, we can see that some samples are above the 75th percentiles. Upon further investigation, we found that the highest two values are in the standard and Dynparity condition and belong to the same user. All other samples are from disjunct users. Only one more sample in the Dynparity condition exceeds the 75th percentile of the standard condition by a large margin. As the answers in the questionnaires are highly subjective, we believe that these outliers are a natural occurrence one can expect in a study with 31 users and that our results are still valid.

## 6.4 | Limitations

Changing the disparity changes the user's vergence, which could change the perceived depth in cases where they focus objects at the screen border. Therefore, a change in depth perception might occur. Furthermore, even if the user does not look at the screen border, Dynparity could also change depth perception for objects in the periphery of the user's field of regard. In total, this also could change the user's overall depth perception. However, Dynparity mainly changes the disparity near the screen border. Previous research shows that the depth perception in the periphery is limited;[21-23] therefore, we speculate that the depth perception near the screen's center is unchanged.

An additional problem of screen-based stereoscopic displays is the accommodation-vergence conflict, which is known to cause visual fatigue.[36] Further studies are needed to investigate whether our method could potentially reduce this type of conflict in the peripheral.

Currently, our implementation runs entirely in the vertex shader. This means that our algorithm does not affect vertices produced in the geometry or tessellation shaders without some extra work. Moreover, triangles that are clipped may lead to an incorrect reprojection close to the borders. In our experience, this will lead to noticeable artifacts only with large triangles, which could be mitigated by adding tessellation. Another issue is that the method currently works on the vertex level. Consequently, our method works best for scenes with a high polygon count and small polygons. In the case of large polygons, our algorithm basically works, but the result could be unexpected. This could be solved by an on-demand tessellation of polygons covering large areas in front of the display.

In this study, we only evaluated the square root function to calculate the new IPD in Equation (4). While our prestudy has shown that linear mapping and one tested sigmoid-function performs worse, other functions, such as the n-root, that further limit the adjustments to the borders could serve better. It remains to be seen whether the reduced distortion in the center of the screen outweighs the increased distortion in the periphery that stems from increased disparity differences between neighboring vertices.

In our study, the user's position was mainly fixed, and there was no interaction. We chose this setup to limit the effects of external factors such as tracking errors and differences in the amount of stereo violations the users would encounter. As Dynparity retains the ability to interact with the virtual environment, as usual, further studies should be conducted to investigate whether Dynparity is beneficial in interactive scenarios.

Lastly, based on the findings from our prestudy, we only considered the distance to the horizontal borders of the screen. In general, we assume a mostly upright head orientation where the eyes naturally align with the horizontal axis. If the user tilts their head 90° around the $z$-axis, there currently is no adjustment at the top and bottom borders. This could be trivially extended to include the vertical distance by considering the closest of all four borders but would increase the amount of distortions. Other, more complex methods could consider the current head rotation. These methods have to investigate potential transfer functions and their range. Their performance has to be evaluated against each other in a future study.

## 7 | CONCLUSION

In this article, we have presented our novel Dynparity stereo projection algorithm that avoids stereo window violations at the borders of stereoscopic screens. Our method runs in real-time as a simple additional step in the vertex shader in a single pass of the standard rendering pipeline. Furthermore, we have implemented Dynparity in the Godot game engine to demonstrate that it can be integrated quite easily.

We evaluated our method and compared it against standard stereoscopic rendering on a $4 \times 2.5\,\mathrm{m}^2$ powerwall with head tracking. In a user study with 31 participants, only two users noticed a slightly visible distortion in the projection of the scenes; also, we found no significant difference in user preference overall. This could suggest that Dynparity does not create any adverse rendering artifacts, but a larger sample size would be needed to confirm this. The evaluation of post-exposure questionnaires revealed significantly lower oculomotor symptoms with our Dynparity rendering compared to conventional stereo projection. This means Dynparity can reduce simulator sickness.

Our approach opens up several avenues for future works. First, we tested only scenes where the disparity appears mainly in the peripheral field of view. In cases where the stereo window violation appears more in the center of interest (because the user is looking closer in the direction of the screen's border), the preference choices could favor Dynparity. Also, with smaller stereoscopic screens, such as the zSpace device, those violations would occur closer to the user's central field of view. In the future, we plan to evaluate our algorithm on the zSpace, a monitor-sized display with head tracking, which is typically at relatively close distances to the user.

Future studies should also investigate the influence of Dynparity rendering on the overall depth perception, especially on hand-eye coordination.

Moreover, the limitations due to the implementation in the vertex shader could be solved by an on-demand tessellation of polygons covering large areas in front of the display.

In general, Dynparity retains the ability to interact with the virtual environment as usual. Users interacting with an object to complete a task usually focus their attention on it.[37] If the user moves the object to the border, Dynparity will prevent stereo window violations and, therefore, we hypothesize that it will increase the user's comfort while interacting. However, this remains to be confirmed with a formal user study.

Finally, the application to setups other than interactive, head-tracked stereoscopic displays could be interesting. For example, even if directors of 3D movies aim to avoid stereo window violations, most 3D TVs offer the possibility of generating (pseudo) 3D images from traditional 2D movies. Maybe, Dynparity could help to improve the quality of this automatically generated 3D video footage.

**ACKNOWLEDGMENTS**
We would like to thank Olaf Schmidt-Kiy from the Fielmann Akademie Schloss Plön for helping us with the study by providing the Titmus Fly Test. Open Access funding enabled and organized by Projekt DEAL.

## ORCID

*Christoph Schröder-Dering* https://orcid.org/0000-0003-0722-9222
*Gabriel Zachmann* https://orcid.org/0000-0001-8155-1127

## REFERENCES

1. López JP, Rodrigo JA, Jiménez D, Menéndez JM. Stereoscopic 3D video quality assessment based on depth maps and video motion. EURASIP J Image Video Process. 2013;2013(1):62.
2. Mendiburu B. 3D movie making: stereoscopic digital cinema from script to screen. 1st ed. New York, NY: Routledge; 2012.
3. Terzić K, Hansard M. Methods for reducing visual discomfort in stereoscopic 3D: a review. Signal Process Image Commun. 2016;47: 402–16.
4. Xu D, Coria LE, Nasiopoulos P. Quality of experience for the horizontal pixel parallax adjustment of stereoscopic 3D videos. Proceedings of the 2012 IEEE International Conference on Consumer Electronics (ICCE); 2012. p. 394–5.
5. Chen MJ, Kwon DK, Cormack LK, Bovik AC. Optimizing 3D image display using the stereoacuity function. Proceedings of the 2012 19th IEEE International Conference on Image Processing. Orlando, FL: IEEE; 2012. p. 617–20.
6. Pritch Y, Ben-Ezra M, Peleg S. Automatic disparity control in stereo panoramas (OmniStereo). Proceedings IEEE Workshop on Omnidirectional Vision (Cat. No.PR00704); 2000. p. 54–61. IEEE Comput. Soc, Hilton Head Island, SC
7. Ide K, Sikora T. Adaptive parallax for 3D television. Proceedings of the 2010 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video; 2010. p. 1–4.
8. Xu J, Yan F, Cao X. Stereoacuity-guided depth image based rendering. Proceedings of the 2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW). Chengdu, China: IEEE; 2014. p. 1–6.
9. Lin HS, Guan SH, Lee CT, Ouhyoung M. Stereoscopic 3D experience optimization using cropping and warping. SIGGRAPH Asia 2011 Sketches. Association for Computing Machinery; 2011.
10. Lang M, Hornung A, Wang O, Poulakos S, Smolic A, Gross M. Nonlinear disparity mapping for stereoscopic 3D. ACM Trans Graph. 2010;29(4):75:1–75:10.
11. Ware C, Gobrecht C, Paton M. Dynamic adjustment of stereo display parameters. IEEE Trans Syst Man Cybern A Syst Humans. 1998;28(1):56–65.
12. Sun G, Holliman N. Evaluating methods for controlling depth perception in stereoscopic cinematography. In: Woods AJ, Holliman NS, Merritt JO, editors. Stereoscopic displays and applications. San Jose, CA: SPIE; 2009. 72370I.
13. Didyk P, Ritschel T, Eisemann E, Myszkowski K, Seidel HP. A perceptual model for disparity. ACM Trans Graph. 2011; 30(4):1–10.
14. Oskam T, Hornung A, Bowles H, Mitchell K, Gross M. OSCAM - optimized stereoscopic camera control for interactive 3D. Proceedings of the 2011 SIGGRAPH Asia Conference, SA '11. Association for Computing Machinery, New York, NY; 2011. p. 1–8.
15. Celikcan U, Cimen G, Kevinc EB, Capin T. Attention-aware disparity control in interactive environments. Vis Comput. 2013; 29(6):685–94.
16. Koulieris GA, Drettakis G, Cunningham D, Mania K. Gaze prediction using machine learning for dynamic stereo manipulation in games. Proceedings of the 2016 IEEE Virtual Reality (VR); 2016. p. 113–20.
17. Avan E, Capin T, Gürçay H, Celikcan U. Enhancing VR experience with RBF interpolation based dynamic tuning of stereoscopic rendering. Comput Graph. 2021;102:390-401.
18. Lou, R., Chardonnet, J.R.: Reducing cybersickness by geometry deformation. Proceedings of the 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR); 2019. p. 1058–9.
19. Lou R, So RHY, Bechmann D. Geometric deformation for reducing optic flow and cybersickness dose value in VR. In: Sauvage B, Hasic-Telalovic J, editors. Eurographics 2022 - Posters. Switzerland: The Eurographics Association; 2022.
20. Bruder G, Argelaguet F, Olivier AH, Lécuyer A. CAVE size matters: effects of screen distance and parallax on distance estimation in large immersive display setups. Presence Teleoperat Vir Environ. 2016;25(1):1–16.
21. Wardle SG, Bex PJ, Cass J, Alais D. Stereoacuity in the periphery is limited by internal noise. J Vis. 2012;12(6):12.
22. Mochizuki H, Shoji N, Ando E, Otsuka M, Takahashi K, Handa T. The magnitude of stereopsis in peripheral visual fields. Kitasato Med J. 2012;42:5.
23. Devisme C, Drobe B, Monot A, Droulez J. Stereoscopic depth perception in peripheral field and global processing of horizontal disparity gradient pattern. Vision Res. 2008;48(6):753–64.
24. Kooima R. Perspective projection for VR. In: Sherman WR, editor. VR developer gems. New York, NY: A K Peters/CRC Press; 2019.
25. International telecommunication union: subjective methods for the assessment of stereoscopic 3DTV systems. Recommendation ITU-R BT BT.2021; 2012.
26. Kennedy RS, Lane NE, Berbaum KS, Lilienthal MG. Simulator sickness questionnaire: an enhanced method for quantifying simulator sickness. Int J Aviat Psychol. 1993;3(3):203–20.
27. Bouchard S, Robillard G, Renaud P. Revising the factor structure of the simulator sickness questionnaire. Annu Rev CyberTherapy Telemed. 2007;5:128–37.
28. Hirzle T, Cordts M, Rukzio E, Gugenheimer J, Bulling A. A critical assessment of the use of SSQ as a measure of general discomfort in VR head-mounted displays. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. ACM, Yokohama Japan; 2021. p. 1–14.

29. Krosnick JA, Presser S. Question and questionnaire design. In: Wright JD, Marsden PV, editors. Handbook of survey research. 2nd ed. Cambridge: Academic Press; 2009. p. 1432–33.
30. Bimberg P, Weissker T, Kulik A. On the usage of the simulator sickness questionnaire for virtual reality research. Proceedings of the 2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW); Atlanta, GA: IEEE; 2020. p. 464–7.
31. Deutschen Ophthalmologischen Gesellschaft, Berufsverband der Augenärzte Deutschlands: Empfehlung der Deutschen Ophthalmologischen Gesellschaft und des Berufsverbandes der Augenärzte Deutschlands zur Fahreignungsbegutachtung für den Straßenverkehr; 2008.
32. Taylor RM, Hudson TC, Seeger A, Weber H, Juliano J, Helser AT. VRPN: a device-independent, network-transparent VR peripheral system. Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST '01. Association for Computing Machinery; 2001. p. 55–61
33. King DE. Dlib-ml: a machine learning toolkit v.19.22. J Mach Learn Res. 2009;10:1755–8.
34. Quade D. Using weighted rankings in the analysis of complete blocks with additive block effects. J Am Stat Assoc. 1979;74(367): 680–3.
35. Theodorsson-Norheim E. Friedman and Quade tests: BASIC computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples. Comput Biol Med. 1987;17(2):85–99.
36. Hoffman DM, Girshick AR, Akeley K, Banks MS. Vergence–accommodation conflicts hinder visual performance and cause visual fatigue. J Vis. 2008;8(3):33.
37. Smith TJ, Mital PK. Attentional synchrony and the influence of viewing task on gaze behavior in static and dynamic scenes. J Vis. 2013;13(8):16.

## AUTHOR BIOGRAPHIES

**Christoph Schröder-Dering** received his M.Sc. degree in informatics from the University of Lübeck, Germany, in 2015. He is currently pursuing a Ph.D. at the Computer Graphics and Virtual Reality group of the University of Bremen. His research interests include VR projection techniques and machine learning on eye-tracking as well as 3D data.

**Gabriel Zachmann** is professor for computer graphics, visual computing, and virtual reality at University of Bremen, Germany, since 2012. Before that, he established and headed the computer graphics group at Clausthal University, Germany, where he was a professor with the Computer Science Department since 2005. In 2000, Dr. Zachmann received a Ph.D. in computer science, and in 1994 a Dipl.-Inform (M.Sc.), both from Darmstadt University. His research interests are virtual reality, geometric computing, massive parallel algorithms, and medical virtual simulations.

**René Weller** received his Ph.D. in Computer Science in 2012 from the University of Bremen. Currently, he is a postdoctoral research assistant at the Computer Graphics Group at the University of Bremen and works in the field of computer graphics and virtual reality with specialization in geometric algorithms for collision detection, sphere packings and haptic rendering.

**How to cite this article:** Schröder-Dering C, Zachmann G, Weller R. Dynparity: Dynamic disparity adjustment to avoid stereo window violations on stationary stereoscopic displays. Comput Anim Virtual Worlds. 2022;e2099. https://doi.org/10.1002/cav.2099

## APPENDIX

```
in vec4 vertex_attrib;

uniform SceneData {
  // asymetric projection matrix
  mat4 projection_matrix;
  // moves the camera into origin
  mat4 camera_inverse_matrix;
  // screen corners in world space
  vec3 p_a, p_b, p_c;
  // eye position and offset incl. rotation
  vec3 pe, eye_offset;
  float near, far;
}

void fill_projection(vec3 sa, sb, sc, pe
    float n, float f, out mat4 m) {
  // Fills m with normal stereo projection
  // for a screen defined by sa, sb, and sc.
  // The camera is at pe and n, f are the
  // near and far plane distances
}

void main() {
  // standard projection
  MV = camera_inv_matrix * world_matrix;
  vertex = MV * vertex_attrib;
  gl_Position = proj_mat * vertex;

  // Dynparity
  world_v = world_matrix * vertex_attrib;
  z = world_v.z;

  // only change vertices in front of the
  // screen, assume it is in the xy plane
  if (z <= 0.0)
    return;

  // from clip space to NDC (−1,1)
  gl_Position = gl_Position / gl_Position.w;
  border_dist = 1 − abs(gl_Position.x);
  // f()
  border_dist = sqrt(border_dist);

  // for vertices outside of the screen
  border_dist = clamp(border_dist, 0.0, 1.0);

  // compute eye offset
  eye_pos = pe + border_dist * eye_offset;

  // recompute projection_matrix
  fill_projection(p_a, p_b, p_c, eye_pos,
    near, far, out proj_mat);

  // Account for the new eye position as the
  // vertex is already in view space
  T = mat4(1.0);
  T[3].xyz = (1 - border_dist) * eye_offset;
  proj_mat = proj_mat * T;

  // the final vertex position in clip space
  gl_Position = proj_mat * vec4(vertex, 1.0);
}
```

Listing 1: Vertex shader pseudocode