Less Cybersickness, Please: Demystifying and Detecting Stereoscopic Visual Inconsistencies in Virtual Reality Apps

SHUQING LI, The Chinese University of Hong Kong, China
CUIYUN GAO*, Harbin Institute of Technology, China
JIANPING ZHANG, The Chinese University of Hong Kong, China
YUJIA ZHANG, Harbin Institute of Technology, China
YEPANG LIU†, Southern University of Science and Technology, China
JIAZHEN GU, The Chinese University of Hong Kong, China
YUN PENG, The Chinese University of Hong Kong, China
MICHAEL R. LYU, The Chinese University of Hong Kong, China

The quality of Virtual Reality (VR) apps is vital, particularly the rendering quality of the VR Graphical User Interface (GUI). Different from traditional two-dimensional (2D) apps, VR apps create a 3D digital scene for users, by rendering two distinct 2D images for the user's left and right eyes, respectively. Stereoscopic visual inconsistency (denoted as "SVI") issues, however, undermine the rendering process of the user's brain, leading to user discomfort and even adverse health effects. Such issues commonly exist in VR apps but remain under-explored. To comprehensively understand the SVI issues, we conduct an empirical analysis on 282 SVI bug reports collected from 15 VR platforms, summarizing 15 types of manifestations of the issues. The empirical analysis reveals that automatically detecting SVI issues is challenging, mainly because: (1) lack of training data; (2) the manifestations of SVI issues are diverse, complicated, and often application-specific; (3) most accessible VR apps are closed-source commercial software, we have no access to code, scene configurations, etc. for issue detection. Our findings imply that the existing pattern-based supervised classification approaches may be inapplicable or ineffective in detecting the SVI issues.

To counter these challenges, we propose an unsupervised black-box testing framework named StereoID to identify the stereoscopic visual inconsistencies, based only on the rendered GUI states. StereoID generates a synthetic right-eye image based on the actual left-eye image and computes distances between the synthetic right-eye image and the actual right-eye image to detect SVI issues. We propose a depth-aware conditional stereo image translator to power the image generation process, which captures the expected perspective shifts between left-eye and right-eye images. We build a large-scale unlabeled VR stereo screenshot dataset with larger than 171K images from 288 real-world VR apps, which can be utilized to train our depth-aware conditional stereo image translator and evaluate the whole testing framework StereoID. After substantial experiments, depth-aware conditional stereo image translator demonstrates superior performance in generating stereo

*Corresponding author.

†Yepang Liu is affiliated with both the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering at Southern University of Science and Technology.

Authors' addresses: Shuqing Li, The Chinese University of Hong Kong, Hong Kong, China, sqli21@cse.cuhk.edu.hk; Cuiyun Gao, Harbin Institute of Technology, Shenzhen, China, gaocuiyun@hit.edu.cn; Jianping Zhang, The Chinese University of Hong Kong, Hong Kong, China, jpzhang@cse.cuhk.edu.hk; Yujia Zhang, Harbin Institute of Technology, Shenzhen, China, 200110910@stu.hit.edu.cn; Yepang Liu, Southern University of Science and Technology, Shenzhen, China, liuyp1@sustech.edu.cn; Jiazhen Gu, The Chinese University of Hong Kong, Hong Kong, China, jiazhengu@cuhk.edu.hk; Yun Peng, The Chinese University of Hong Kong, Hong Kong, China, ypeng@cse.cuhk.edu.hk; Michael R. Lyu, The Chinese University of Hong Kong, Hong Kong, China, lyu@cse.cuhk.edu.hk.

ACM 2994-970X/2024/7-ART96
https://doi.org/10.1145/3660803

images, outpacing traditional architectures. It achieved the lowest average L1 and L2 losses and the highest SSIM score, signifying its effectiveness in pixel-level accuracy and structural consistency for VR apps. StereoID further demonstrates its power for detecting SVI issues in both user reports and wild VR apps. In summary, this novel framework enables effective detection of elusive SVI issues, benefiting the quality of VR apps.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → **Virtual reality**.

Additional Key Words and Phrases: Automated Testing, Virtual Reality, Extended Reality, Software Quality Assurance, GUI Testing, Deep Learning

ACM Reference Format:
Shuqing Li, Cuiyun Gao, Jianping Zhang, Yujia Zhang, Yepang Liu, Jiazhen Gu, Yun Peng, and Michael R. Lyu. 2024. Less Cybersickness, Please: Demystifying and Detecting Stereoscopic Visual Inconsistencies in Virtual Reality Apps. *Proc. ACM Softw. Eng.* 1, FSE, Article 96 (July 2024), 23 pages. https://doi.org/10.1145/3660803

1 INTRODUCTION

Virtual Reality (VR) is a technology that provides users with immersive experiences by creating interactive virtual environments. Over the past few years, VR has experienced a remarkable surge in popularity and diversity, encompassing tens of thousands of apps [46] tailored for various purposes such as skill training [2], entertainment [1, 3], and even usage scenarios that require high reliability like medical procedures [41]. This successful deployment has captivated a global user base of exceeding 171 million people [66]. VR apps adopt stereoscopic 3D (S3D) [34, 58], which provides two distinct two-dimensional (2D) images for the eyes of the user respectively. The user's brain then constructs the corresponding stereoscopic 3D scene with an illusion of depth based on these two images. Rendering issues can lead to discomfort feelings in VR, which is the well-known cybersickness[1] [30, 51] problem. The cybersickness, including symptoms such as headaches, disorientation, and nausea, potentially affects the users' health and safety, hindering the development and growth of VR apps [43, 47, 53, 54, 57].

A common cause of rendering-induced cybersickness issues is the inconsistent left and right eye view from the 2D-to-S3D construction process. Figure 1 illustrates an example. This issue reported on GitHub [9] pertains to left-eye rendering corruption in the entry-point application for all Steam VR apps, *SteamVR Home* [10]. In the figure, some virtual objects in the left eye present completely inverted colors, while others display washed-out hues. Additionally, the Skybox, which projects the 3D panoramic background scene, appears to be absent in the left-eye view. We refer to such issues on VR apps' GUI as ***Stereoscopic Visual Inconsistencies (SVI)***. The inconsistencies not only mislead users with conflicting information but also discourage users from playing the VR application. Manual playtesting with human testers could be one possible solution to mitigate SVI issues [27], but it is time-consuming, labor-intensive, and may expose testers to health and safety risks [43, 53], prompting researchers to develop automated testing methods.

To better understand the SVI issues in real-world VR apps, we first collect 282 bug reports of SVI issues from 15 VR-related platforms and conduct an empirical analysis to manually analyze their manifestations. Our findings reveal that the SVI issues are diversified in both scale and manifestation, and are closely tied to the semantics of the VR apps' semantics or logic. For scale, the symptoms span both view level and object level. View-level inconsistencies are global and encompass view displacement, deformation, and view angle discrepancies. Object-level inconsistencies are local and related to object quantity, rendering effects, position, etc. For manifestations, we summarize 15 different categories of them. Instead of causing application crashes or runtime errors, these issues only affect user experience and thus can hardly be detected with regular test oracles.

[1] We use cybersickness and VR sickness interchangeably in this paper.

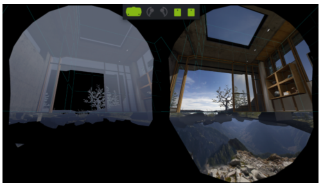


Fig. 1. A bug causing stereoscopic visual inconsistencies in the SteamVR home app (issue #299 of ValveSoftware/SteamVR-for-Linux [9] on GitHub)

Recent research has taken steps to address the above-mentioned shortcomings of conventional test oracles and testing approaches in detecting GUI display issues, using deep learning (DL) techniques in automated GUI testing [32, 49, 50, 72]. These methods model GUI issue detection as a classification problem, augmenting training data by generating abnormal screen captures through modifications of regular GUI screenshots or bug injections into the code. Such supervised approaches enable DL classifiers to detect faults effectively and enhance GUI testing efficiency. Despite the advancements, they still face the following three challenges in identifying SVI issues: (1) *Lack of labeled training data.* It is hard to collect sufficient labeled data for model training, especially data with confirmed SVI issues. Manual inspection of large amounts of data which might cause cybersickness is inapplicable. Data augmentation methods cannot work well, either, due to the following two challenges. (2) *Semantic-related manifestations can hardly be captured by pre-defined patterns.* SVI issues are closely linked to application-specific semantics, making it challenging to be captured using existing pattern-based detection techniques. Besides, current approaches are limited to detecting predefined patterns and can hardly handle unreported symptoms. (3) *Closed-source VR apps provide limited accessible information for issue detection.* Some approaches supplement issue detectors with internal application data like code and scene configurations [32, 50], while commercial VR apps only expose externally rendered states for SVI issue detection.

To address the challenges, in this paper, we propose **StereoID**, an automated testing framework to **ID**entify **Stereo**scopic visual inconsistencies in VR apps. StereoID relies solely on external rendered states of VR apps and does not require additional information such as code configurations. Instead of pre-defining detection patterns, StereoID reformulates the SVI issues identification problem into an anomaly detection problem. StereoID generates a synthetic right-eye image based on the actual left-eye image and computes distances between the synthetic and the actual right-eye images to detect anomaly[2]. For the generation of synthetic right-eye images, we propose a de**P**th-aw**A**re cond**I**tio**N**al s**T**ereo imag**E** translato**R** (**Painter** in short). Painter captures the complicated but predictable mappings between left-eye and right-eye images. To deal with the spatial shift between left-eye and right-eye images due to object depths in the scene, Painter integrates monocular depth maps for the left-eye image and the right-eye image respectively as additional inputs. This depth-aware manner empowers Painter with the crucial spatial context it requires to generate accurate right-eye images.

In summary, we make the following contributions:

- To the best of our knowledge, our work is the first to systematically analyze and detect the stereoscopic visual inconsistencies (SVI issues) in real-world VR apps. We build a dataset of bug reports and screenshots with SVI issues.
- We construct a large-scale dataset of over 171K VR stereo image screenshots via execution of 288 real-world VR apps.

[2] The model can be used to generate from left to right and vice versa. We take the left-right direction as an example in the rest part of the paper.

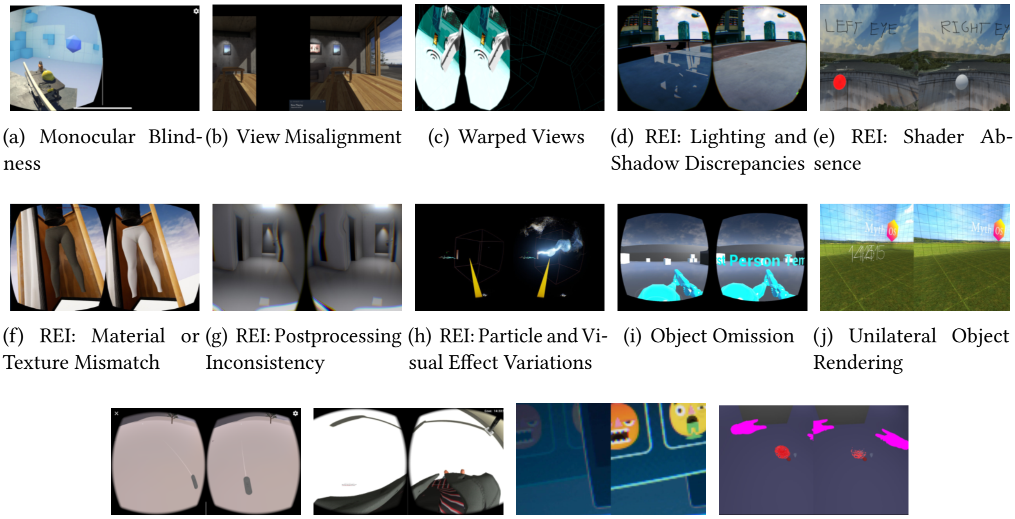
- We propose StereoID, an automatic testing framework to detect SVI issues. StereoID is empowered by a novel depth-aware conditional stereo image translator. Extensive evaluations show that StereoID can effectively detect SVI issues in real-world VR apps.
- To facilitate follow-up studies, we release our datasets and StereoID, at https://sites.google.com/view/stereoid.

2 EMPIRICAL ANALYSIS OF SVI ISSUES

2.1 Data Collection

We collect real-world SVI issues by searching the keywords in reports of VR users or developers from 15 platforms: VR-related online forums and app stores. The 15 platforms include VR online forums (Unity-related forums [16, 17, 19], Unreal Engine related forums [7, 22]), VR app stores and app store forums (Meta Quest App Store [5], Meta Quest App Lab [4], VIVEPORT [25], SideQuest [6], Steam [26], Meta Community Forums [11], VIVE Forum [24], Steam Community [15]), GitHub [8], and Stack Overflow [14]. To include as many related posts as possible, we start by sampling some posts and analyzing the related keywords, such as *eye render, rendering left eye, rendering right eye, two eyes, both eyes, eyes render difference, inconsistent render (display)*, in the sampled posts. The identified keywords are then used to search for more related posts for keyword analysis. This process iterates until no more new posts and keywords can be found. After the keyword search, we get 3,266 candidate bug reports. As keyword search cannot guarantee that the candidate bug reports are related to SVI issues, two authors further check the contents of candidate bug reports. 282 distinct bug reports are agreed to be related to SVI issues by both authors and we finally collect a screenshot dataset of 108 image pairs with real-world SVI issues guided by the 282 bug reports.

2.2 Categorizing the Manifestation of SVI Issues



(a) Monocular Blindness (b) View Misalignment (c) Warped Views (d) REI: Lighting and Shadow Discrepancies (e) REI: Shader Absence

(f) REI: Material or Texture Mismatch (g) REI: Postprocessing Inconsistency (h) REI: Particle and Visual Effect Variations (i) Object Omission (j) Unilateral Object Rendering

(k) Object Position Discrepancy (l) Object Warping (m) Level of Detail Inconsistency (n) Partial Object Rendering

Fig. 2. Examples of stereoscopic visual inconsistencies

Following the widely-adopted open coding procedure [33], two authors of this paper, with over three years of software development experience and sufficient domain knowledge on VR, perform further analysis on the manifestation of SVI issues. They first individually examine the title, description, discussions, and all uploaded attachments, such as screenshots, of the collected bug reports to understand the specific manifestation of each SVI issue, and then discuss their results to add/delete, update, or merge codes together. In case of disagreement in the discussion, another three authors are involved in making final decisions. Finally, we reach a consensus on the categorization of SVI issues and report our findings as follows[3].

The SVI issues are classified into two types: view-level (global) and object-level (local) inconsistencies. View-level inconsistencies refer to issues that impact the entire view, while object-level inconsistencies pertain to inaccurate rendering of single objects. Note that some bug reports have multiple manifestations and thus they belong to multiple categories. We classify bug reports with vague descriptions and manifestations with low occurrences into the "Other" category.

We present the refined symptom categories of SVI issues in the rest of the section.

2.2.1 *View-Level Inconsistencies.*

Monocular Blindness (17%). This category covers issues where one eye fails to render any visual information, resulting in a blank or black screen for that eye. Figure 2(a) shows that the right-eye image is not rendered.

View Misalignment (6%). This category pertains to issues where the left-eye and right-eye views are misaligned, causing a noticeable offset between the two views. Figure 2(b) depicts the left-eye view as distinctly offset to the left, obscuring its full view for the user.

Warped Views (3%). This category includes issues where the visual information displayed to each eye is distorted or stretched, resulting in a warped or unnatural view. Unlike view misalignment, warped views can also cause object distortion and alter the overall proportions of the view. Figure 2(c) shows the left-eye and right-eye images rendered normally but with abnormal compression and ratio. Ultimately, only the left-eye view is displayed, leaving the right-eye view dark.

Asymmetric Viewing Angles (1%). This category involves issues where the viewing angles of the left and right eye views are inconsistent, or one eye's view is flipped or skewed. For example, one bug report [21] shows the following manifestations: "*when each eye gazes in a different direction, the left eye sees the object from the side while the right eye sees it from the front.*" This difference in perspective, despite their similar positions in the field of view, leads to double vision and dizziness.

2.2.2 *Object-Level Inconsistencies.*

Rendering Effect Inconsistencies (REI) (29%). This category covers various types of inconsistencies related to the rendering of graphical effects as follows.

- *Lighting and Shadow Discrepancies (11%)*: differences or absences in lighting, shadows, or reflections between the eye views. In Figure 2(d), there is an inconsistency in the shading between the left and right eyes.
- *Shader Absence (6%)*: missing or incomplete visual effects in one eye view due to shaders not rendering. In Figure 2(e), a user-defined shader for a red highlight material is rendered solely in the left eye, while being absent in the right eye.
- *Material or Texture Mismatch (6%)*: missing or mismatched textures between the two eyes. In Figure 2(f), the materials of clothing for the left and right eyes differ, with the left eye featuring black pants and the right eye featuring white ones.
- *Post-Processing Inconsistency (3%)*: inconsistent post-processing effects between the two eyes. Post-processing effects offer several specific rendering effects with little latency for developers [12, 13]. Inconsistent post-processing effects can lead to vignettes, blurring, or ghosting in the monocular view. In Figure 2(g), there is a disturbance and blurriness in the center of the left and right eye views, as they appear at different positions.

[3] The raw data can be found in our dataset.

- *Particle and Visual Effect Variations (3%).* The Visual Effect Graph [23] creates a particle system to simulate particle behavior, generating Visual Effects like varying appearances, explosions, or smoke within a single view, greatly enhancing immersion and gameplay. Particle effects can be considered specialized effects generated by a particle system. Inconsistency of such effects can cause variations in phenomena such as smoke, sparks, shooting stars, clouds, dust, and more in both eyes. For instance, in Figure 2(h), GPU particles in the view are only rendered on the right eye, leaving the corresponding part of the left eye black.

Object Omission (20%). This category pertains to issues where some objects are not rendered in either eye view, leading to a loss of important visual information. E.g., in Figure 2(i), the blue text is only visible in the right eye, indicating the omission of the text object in the left eye view.

Unilateral Object Rendering (7%). This category includes repeated objects or foreign elements that shouldn't exist in the monocular view, leading to an uneven visual experience for the user. For example, the time text shown in Figure 2(j) is repeated in the left eye.

Object Position Discrepancy (7%). This category pertains to issues where the position or orientation of objects differs between the left and right eye views. For instance, in Figure 2(k) the controller appears in a different position in each eye.

Object Warping (2%). This category encompasses issues where some objects appear distorted or stretched in one eye view. In Figure 2(l), the shadow looks bigger in the left image and the jacket is stretched.

Level of Detail (LOD) Inconsistency (2%). The level of detail refers to the complexity of 3D VR models which can be decreased for distant or dynamically changing objects. This category includes issues where the level of detail differs between the two eyes' views, such as level of detail inconsistencies or anti-aliasing problems. In one bug report from the Unity Forum [20], the VR application displays different levels of detail for each eye, leading to disparities at specific distances. For example, in Figure 2(m), when two planes intersect in front of the user, it creates an erroneous perception of varying distances between the planes and the L/R perspectives, resulting in incorrect depth perception for users.

Partial Object Rendering (1%). This category involves issues where some objects are only partially rendered. Take Figure 2(n) as an example, the red fluid is displayed normally in one eye, but becomes mutilated and incomplete in the other eye [18].

2.3 Challenges to Automatic Detection of SVI Issues

SVI issues adversely affect user experience without causing application crashes or throwing errors, making the detection of them difficult with regular test oracles. There exist prior studies [32, 49, 50, 72] on automated GUI test oracles that employ DL techniques for identifying abnormal GUI states in mobile, web apps, and games. These methods typically formulate the process of GUI issue detection as a classification task. To enhance the training data, they generate anomalous screen captures by modifying standard GUI screenshots or by injecting bugs directly into the codebase. Further details are presented in Section 7. However, despite these progressive advancements, the existing methods struggle to accurately identify SVI issues and face the following challenges.

Challenge 1: Lack of training data. While the concerns of SVI issues widely exist in the forums of VR apps, few bug reports from users include the described problematic images. This is evidenced by only 108 (<40%) image pairs extracted from the 282 bug reports. For data augmentation methods to enrich the dataset, both image-based and code-based methods rely on manually identified glitch or bug patterns, which exhibit similar manifestations across different apps, such as random noise, overexposure, and black borders. However, as introduced in Section 2.2, the symptoms of SVI issues

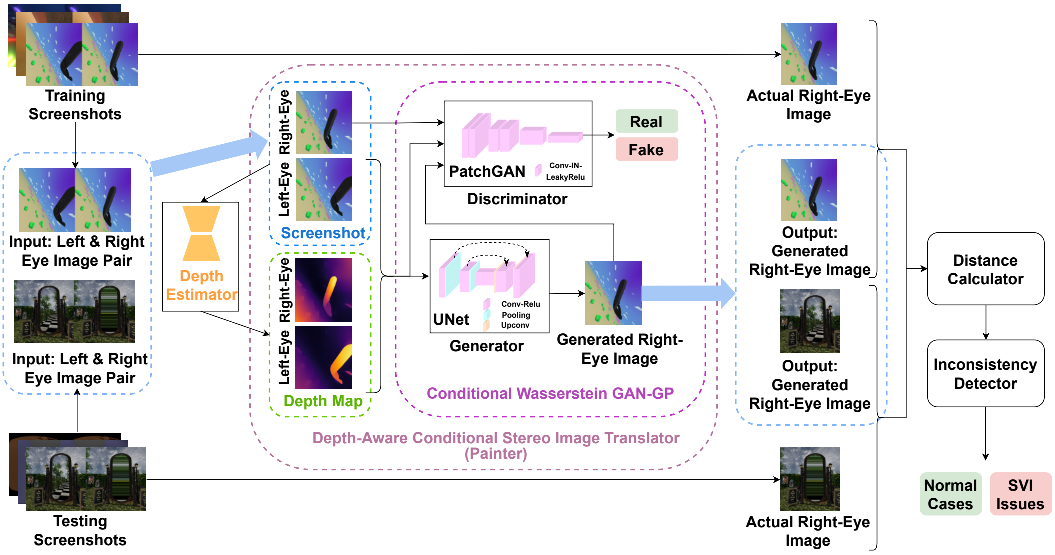


Fig. 3. Overview of StereoID

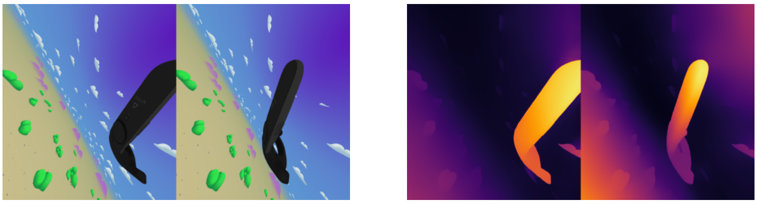
are strongly related to application semantics and exhibit diverse manifestations across different scenarios, making it challenging to capture them with a set of general image patterns.

Challenge 2: Semantic-related manifestations can hardly be captured by pre-defined patterns. To augment with, neither image-based nor code-based data augmentation can work well even on one specific image pattern. For image-based methods, complex S3D rendering for VR is hard to manipulate and mimic feasibly according to a predefined image pattern as real-world SVI issues, while maintaining the consistency of the rest part. We observe that the root cause of SVI issues is more complex than several lines of code changes, and the bugs arise from multiple sources spanning apps, developing engines, runtime, and related libraries. Thus, it is also hard to inject bugs and conduct code-based data augmentation.

Challenge 3: Closed-source VR apps provide limited accessible information for issue detection. Previous approaches require semantics information derived from internal application data such as code, assets, and scene configurations. The internal application data, however, can hardly be accessed for commercial VR apps. Limited input information obtained from commercial VR apps hinders previous approaches to effectively detect SVI issues in practice.

3 APPROACH: STEREOID

In this section, we elaborate on the proposed automatic testing framework, named StereoID, for identifying SVI issues. As illustrated in Fig. 3, StereoID mainly includes four components, including *monocular depth estimator*, *depth-aware left-right-eye image translator*, *distance calculator*, and *inconsistency detector*. Given a dataset of paired stereo screenshots with left-eye and right-eye images, the *depth-aware conditional stereo image translator* tries to learn the mapping relations between the rendered GUI screenshots of the two eyes to generate synthetic right-eye images given the left-eye image. To make the generation a depth-aware process to calculate and involve the 2D representations (referred to as depth maps) that encode the distance between the viewer and virtual objects in the scene for each pixel for both eyes. A distance metric is then computed by the *distance calculator* between the generated synthetic right-eye image and the real right-eye image, which serves as the basis for the inconsistency detection procedure. Outliers in this metric



(a) Original stereoscopic screenshot (left: left-eye view, right: right-eye view)

(b) Depth map generated by monocular depth estimator (Section 3.1)

Fig. 4. Illustration of the stereoscopic screenshot and the corresponding depth map[4]

space are finally identified by *inconsistency detector* as SVI issues. The StereoID framework is unsupervised and does not need any annotated SVI issues.

3.1 Monocular Depth Estimator

In StereoID, we consider the depth information inherent in the stereo-mapping generation process in S3D VR scenarios. As shown in Fig. 4(a), the relationship between the left-eye and right-eye images essentially relates to the shifts of virtual objects on the scene, and the shift is strongly related to the *depth* of each object (i.e., the distance between the object and the camera/viewer). To accurately synthesize the right-eye image given the left-eye image, we propose to involve depth maps as spatial context input. Specifically, we design the monocular depth estimator component for computing the depth maps of left-eye and right-eye images, respectively, before image generation.

The proposed component incorporates $BEiT_{512} - L$ [60], a robust monocular relative depth estimation model. $BEiT_{512} - L$ employs a Transformer-based architecture which enables it to learn and extract deep features from input images. The model has been trained with multi-objective optimization for image classification on up to 12 datasets. This allows the accurate generation of depth maps, even when the depth varies across a wide range in a single scene. Fig. 4(b) illustrates the depth maps generated by our monocular depth estimator for Figure 4(a), in which each depth map is generated from one monocular image.

3.2 Depth-Aware Conditional Stereo Image Translator

3.2.1 Overview of Painter. With the depth maps generated by the monocular depth estimator, depth-aware conditional stereo image translator (Painter in short) is able to learn the complex mappings from the provided context and generate an expected corresponding stereoscopic image from a monocular-eye view.

The principal architecture of depth-aware conditional stereo image translator is a conditional variant of Generative Adversarial Networks (GAN) [52], specifically a Conditional Wasserstein GAN [29] with Gradient Penalty [37], to generate right-eye images from corresponding left-eye images and their respective depth maps generated by the monocular depth estimator.

The model may not be stable, i.e., sometimes it generates low-quality samples or fails to converge. The gradient penalty (GP) [37] is proposed to enhance the stability of training for the Wasserstein GAN framework. To ensure the stability of the training process, our model employs a gradient penalty that is designed to penalize the norm of the gradient of the output of the discriminator D with respect to its input. This mechanism encourages the gradient norm to be close to 1, effectively enforcing the Lipschitz constraint without the potential drawbacks of weight clipping. This approach

[4] The depth map assigns each pixel color based on the actual distance from the viewer (camera) to the object, with surfaces closer to the viewpoint appearing darker in color, and those farther away appearing brighter [40].

ensures a stable learning process and precludes the model from potential divergence, thereby increasing the overall robustness of StereoID.

3.2.2 The Generator Architecture of Painter. In our proposed model, the synthesis of the right-eye images is carried out by a generator constructed around a U-Net architecture [62]). This architecture is a prominent choice in image synthesis due to its capacity to capture intricate details and retain context through its design. The U-Net incorporates an encoder-decoder framework complemented by skip connections between mirrored layers in the encoder and decoder blocks. This unique configuration facilitates the transfer of low-level, localized information directly across the network, thereby enhancing the quality of the synthesized output.

The generator is structured to accept three inputs: a left-eye image and depth maps of both eyes. These inputs are aggregated along the channel dimension, resulting in a nine-channel tensor which constitutes the initial input to the U-Net.

The architectural design of the generator includes numerous layers. Each layer is composed of two 3×3 convolutions, each succeeded by a Rectified Linear Unit (ReLU) activation function, and a 2×2 max pooling operation with a stride of 2 for downscaling. This structure persists until the model reaches the lowest resolution point in the encoding path.

Following this, the decoding path, or upscaling process, is initiated. Each step in this phase includes an up-convolution operation, subsequent concatenation with the correspondingly deep feature map from the encoding path, and a pair of 3×3 convolutions, each succeeded by a ReLU. The upscaling process continues until an output image of equivalent dimensions to the input left-eye image is generated, representing the synthetic right-eye image.

The number of features per layer in the generator is governed by a hyperparameter denoted as `ngf`, which in our implementation is set to 64. As the network transitions from the encoding to the decoding path, this parameter value doubles and halves respectively at each layer, ensuring a balanced distribution of features across the network.

3.2.3 The Discriminator Architecture of Painter. The underlying discriminator in our conditional Wasserstein generative adversarial network harnesses a Markovian architectural design, which is known as PatchGAN [39]. This specialized architecture operates on segments of the image, specifically patches of dimensions $N \times N$, enabling it to independently classify each patch as either real or synthetic. This patch-based approach bolsters the capability of the network to synthesize images exhibiting sharper details and textures, thereby focusing on localized structural nuances as opposed to a global perspective.

Our discriminator is designed to accept a twelve-channel input: a composition of the nine-channel tensor provided to the generator (comprising a left-eye image and two depth maps), in addition to the three-channel output from the generator, representing the synthetic right-eye image. Each discriminator input corresponds to a 70×70 patch of an image.

The structure of the discriminator is characterized by a series of layers, with each layer composed of a Convolution-BatchNorm-LeakyReLU sequence. This number of layers is preset to three in our implementation, and the LeakyReLU activation function employs a slope parameter of 0.2 for negative input values. The number of features per layer commences at 64 and subsequently doubles after each layer.

In the context of enhancing model learning stability, our design incorporates Wasserstein distance [29] coupled with a gradient penalty [37]. This mechanism ensures unit norm gradients, thereby fostering robust learning dynamics and alleviating prevalent issues such as mode collapse often encountered in conventional GAN designs. The magnitude of this gradient penalty is modulated by a hyperparameter, λ_{gp}, which is preset to a value of ten, adhering to widely adhered standards.

The final aspect of the discriminator architecture revolves around maintaining equilibrium in the training dynamics between the generator and the discriminator. This balance is governed by the *critic_iterations* parameter, specifying the ratio of discriminator iterations per generator iteration. Set to a value of 5 in our model, this parameter ensures that the critic is adequately trained prior to each update in the generator, thereby upholding a balanced training regimen.

3.2.4 Objective Functions and Training Approach. Our model employs a conditional Wasserstein GAN [29] with gradient penalty [37] for the synthesis of right-eye images conditioned on the left-eye image and respective depth maps.

The training methodology necessitates a cyclic optimization of the generator and discriminator with the aim of minimizing an ensemble of loss functions. These functions include the Wasserstein GAN loss (L_{WGAN}), the L1 loss (L_1), and a weighted Mean Squared Error (MSE) loss (L_{WMSE}).

Wasserstein GAN Loss with Gradient Penalty. The L_{WGAN} is designed to quantify the distance between the distributions of authentic and synthesized images. Incorporation of the gradient penalty facilitates the enforcement of Lipschitz continuity, yielding stability to the GAN training procedure. The loss is calculated for both real and generated images.

$$L_{WGAN} = \mathbb{E}_{x\sim P_{\text{real}}}[D(x)] - \mathbb{E}_{z\sim P_z}[D(G(z))] + \lambda E_{\hat{x}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2], \tag{1}$$

where P_{real} and P_z represent the data and generator distributions, D is the discriminator, and G is the generator. $\hat{x}$ denotes a randomly sampled point along the straight line between a real and a generated data point, λ is a hyperparameter that controls the penalty strength, and $|| \cdot ||_2$ is the Euclidean norm.

L1 Loss. The $L1$ loss encourages the generated right-eye images to approximate the ground truth in the absolute difference sense.

$$L_1 = \mathbb{E}_{x,y\sim P_{\text{data}}}[|y - G(x)|_1], \tag{2}$$

where x and y are the input and corresponding target images.

Weighted MSE Loss. The L_{WMSE} loss computes a weighted pixel-level squared difference between the real and generated right-eye images.

$$L_{WMSE} = \mathbb{E}_{x,y\sim P_{\text{data}}}[w(x,y) \cdot (y - G(x))^2], \tag{3}$$

where $w(x,y)$ indicates the weight, which is determined by the absolute difference between the real and generated images.

$$w(x,y) = \frac{1}{1 + e^{-|y-G(x)|_1}}. \tag{4}$$

Optimization Procedure.

The total loss L for the generator is defined as:

$$L = L_{WGAN} + \alpha L_1 + \beta L_{WMSE}, \tag{5}$$

where α and β are hyperparameters that modulate the contribution of each loss term. The model employs the Adam optimizer, iterating between updating the generator and the discriminator.

$$G = \arg\min_G L(G,D), \ D = \arg\max_D L(G,D), \tag{6}$$

where G and D represent the optimal generator and discriminator parameters.

3.3 Inconsistency Detection

From the empirical analysis, we find that SVI issues exhibit a wide array of manifestations, some of which may be unknown or unexpected, making it infeasible to learn the "wrong" patterns. Additionally, SVI issues are naturally rare compared to normal behavior, leading to a scarcity of positive samples for supervised training. However, SVI issues are expected to inhabit less dense regions of the feature space and have distinctive attribute values. Therefore, we formalize the whole testing problem as an anomaly detection process, which focuses on learning the "right" patterns and flagging deviations from these patterns as potential issues.

3.3.1 Distance Calculator. The objective of our inconsistency detector is to identify the most discrepant pairs between synthetic and runtime right-eye images. These pairs serve as indicators of significant inconsistencies between the two image types. To quantify the discrepancy, we employ three well-established image similarity metrics: L1 norm, L2 norm, and the Structural Similarity Index Measure (SSIM). Let I_{syn} represent the synthetic right-eye image and I_{run} denote the runtime right-eye image. The three measures are defined as:

$$L1(I_{syn}, I_{run}) = \sum_{i} |I_{syn}[i] - I_{run}[i]| \tag{7}$$

$$L2(I_{syn}, I_{run}) = \sqrt{\sum_{i} (I_{syn}[i] - I_{run}[i])^2} \tag{8}$$

$$SSIM(I_{syn}, I_{run}) = \frac{(2\mu_{I_{syn}}\mu_{I_{run}} + c_1)(2\sigma_{I_{syn}I_{run}} + c_2)}{(\mu^2_{I_{syn}} + \mu^2_{I_{run}} + c_1)(\sigma^2_{I_{syn}} + \sigma^2_{I_{run}} + c_2)} \tag{9}$$

Where μ denotes the average, σ^2 is the variance, $\sigma_{I_{syn}I_{run}}$ is the covariance of I_{syn} and I_{run}, and c_1, c_2 are constants to stabilize the division with a weak denominator.

3.3.2 Inconsistency Detector. Introducing hyperparameters α, β, and γ, the weighted aggregate discrepancy is obtained by summing these measures:

$$D(I_{syn}, I_{run}) = \alpha \cdot L1(I_{syn}, I_{run}) + \beta \cdot L2(I_{syn}, I_{run}) + \gamma \cdot (1 - SSIM(I_{syn}, I_{run})) \tag{10}$$

The hyperparameters α, β, and γ can be fine-tuned to reflect the importance of each metric in the context of the application. StereoID uses Isolation Forest [48] to perform the inconsistency detection based on $D(I_{syn}, I_{run})$. During detection, StereoID constructs numerous random binary decision trees as isolation trees. Each tree is constructed by recursively partitioning the data based on a randomly selected feature and a random split value between the maximum and minimum values of the selected feature. The anomalies are isolated early, meaning they are closer to the root of the tree, as they are distinct and inhabit less dense regions. Therefore, they have shorter paths on average in these trees compared to normal data points. The length of this path, averaged over multiple trees, forms the anomaly score of a data point. There exist two key hyperparameters during detection, (1) how many trees are constructed to form the Isolation Forest (*n_estimators*), and (2) the proportion of outliers in the dataset (*contamination*), which is used to define the threshold for separating outliers from normal observations. This methodology ensures a structured and comprehensive approach to identifying inconsistencies.

4 EXPERIMENT DESIGN

4.1 Research Questions

Our experiment is designed to answer the following three research questions:

- **RQ1 (Stereo-Mapping Generation):** How effective is our proposed depth-aware conditional stereo image translator in generating stereo images, i.e., generating right-eye images from left-eye images of VR apps?
- **RQ2 (Detecting User-Reported SVI Issues):** How effective is our proposed StereoID in detecting user-reported SVI issues?
- **RQ3 (Detecting Wild SVI Issues):** How well does StereoID perform in detecting SVI issues in wild real-world VR apps?

4.2 Dataset Construction

Since there are no existing stereoscopic datasets for VR apps. The three experimental datasets are entirely collected by ourselves from the following sources.

The first dataset is automatically collected on 288 real-world VR apps. Based on the interaction simulation and automated testing framework for spatial computing extended reality applications proposed by Li et al. [45], we instantiate ten automated testing agents that simulate HTC VIVE device models and perform automated app interactions on ten Windows PCs. The interaction actions are generated randomly. During the automated execution, the testing agents open the *VR View* of two eyes (side-by-side) on the PC and capture screenshots regularly. The 288 VR apps are sampled from 4,215 free *VR Only* apps available on Steam [26], which is one of the most popular VR app stores currently [46]. To ensure the diversity and representativeness of our dataset, we employ a stratified random sampling strategy, selecting Virtual Reality (VR) apps based on their associated categories (tags) within the Steam platform. We initially select one random application from each category to capture the broadest possible scope of VR experiences. Following this initial sampling, we proceed to randomly sample an additional 2% of the apps within each category. At last, we collect 171,740 stereoscopic screenshots as shown in Figure 4(a). Table 1 demonstrates the statistics among manifestation categories in Section 2.2.

The second dataset is composed of screenshots of real-world bug reports collected during the empirical analysis introduced in Section 2. Removing the user-edited screenshots with coverings (e.g., texts), screenshots with only monocular views, partial screenshots, etc. leaving us a buggy dataset of 82 stereoscopic screenshots (or left-right image pairs).

For ease of quantitative evaluation, we construct the third dataset. It is a labeled dataset. We randomly sample a subset from the first dataset, comprising 4,000 VR screenshots. We hire three VR users to rigorously analyze and label these images. All of them are equipped with real VR devices (device model: Pico 4 Pro), using head-mounted displays to view the image in the S3D mode. Each user independently examines the screenshots and labels those stereo-inconsistent images that make them feel cybersickness as SVI issues. To ensure objectiveness, we only consider screenshots that have been marked by all three users as the final set of SVI issues, other screenshots are regarded as normal cases. After manual inspection, we get 237 SVI issues. We further classify these SVI issues into the 14 manifestation categories as introduced in Section 2.2. Table 1 demonstrates the statistics.

4.3 Implementation Details and Experimental Setup

We implement the Painter in Python using the PyTorch framework [56]. The U-Net architecture of the generator is with 256 feature maps. The model is trained by Adam optimizer [42] over batches of 64 input samples with an initial learning rate of 0.001. The activation functions we use are ReLU (for the generator) and LeakyReLU (for the discriminator) [71]. We resize the binocular images to 1024 * 576. During training, we scale the input size of monocular images to 512 pixels in width before performing a random crop to a 512 * 512 dimension. To ensure normalization at each layer of the network, we adopt instance normalization. For the model used in-depth estimation, we directly

Table 1. Manifestation category statistics of SVI issues in our datasets

	Dataset Collected from Online Bug Reports	Sampled Screenshot Dataset
Object Omission	16	44
Lighting and Shadow Discrepancies	11	0
Object Position Discrepancy	9	105
Shader Absence	6	0
Monocular Blindness	5	17
Particle and Visual Effect Variations	5	0
Unilateral Object Rendering	5	7
Material or Texture Mismatch	5	0
Post-Processing Inconsistency	5	0
Level of Detail Inconsistency	4	0
Object Warping	4	21
View Misalignment	3	0
Partial Object Rendering	1	6
Warped Views	1	0
Asymmetric Viewing Angles	0	37
Other	2	0
Total	82	237

use the pre-trained model from the MiDaS v3.1 model suite[60]. For experiments, we split the first screenshot dataset into training, validation, and testing sets with a ratio of 90%, 5%, 5%, resulting in 154,566, 8,587, and 8,587 screenshots in each group. Due to the limit of computation resources, we randomly sample a subset of the training set containing 20,000 screenshots, instead of training Painter on the whole training set. Experiments for RQ1 use the aforementioned subset (20,000) for training, the original validation set (8,587) for validation, and the original test set (8,587) for testing. To detect user-reported SVI issues, we need to put them into a complete testing set with both positive samples and negative samples. To make the results align with real-world scenarios, we align the SVI issue ratio of the whole testing set for RQ2 with the natural SVI issue ratio in real-world data distribution. As demonstrated in Table 1, our manual analysis results show that the natural ratio, i.e., "SVI issues:the whole set" is close to "*237:4,000*". We sample the appropriate number (i.e., 1,302) of the manually-verified normal data (see Section 4.2) and mix them with user-reported SVI issues (total number: 1,384), forming a testing set under the real-world ratio to evaluate RQ2. For RQ3, the experiments are conducted on the manual-labeled dataset (see Section 4.2). There are several key hyperparameters for Inconsistency Detector (see Section 3.3.2). For *contamination* (i.e., 1 - threshold) and *n_estimators*, we randomly split the manual-labeled dataset into a "training set" and a test set (6:4) for parameter tuning. Guided by literature, we conduct a grid search using the metrics of F-1 score for detecting SVI issues for 100 potential *contamination* values from 0.01 to 0.1 evenly spaced, and *n_estimators* from 50 to 300 with a step of 5. As shown in Figure 5, StereoID reaches best results on the F-1 score of detecting SVI issues when *contamination* equals 0.058 (i.e., threshold equals 0.942), *n_estimators* equals 110. Subsequent experiments on RQ2 and RQ3 are conducted using these hyperparameters. Other parameters use default or recommended values, e.g., we set α, β, and γ of Inconsistency Detector as 1.

4.4 Baselines

4.4.1 For Stereo Generation of Painter. To assess the effectiveness of our proposed Painter, we compare it against five well-established baselines that are widely used: three in the image generation task and two in the 2D to 3D image conversion task. All five baselines use default or recommended configurations and hyperparameters from the original papers.

- **ResNet-Based Autoencoder [69].** A simple yet powerful neural network structure for image reconstruction.
- **U-Net** [62]. A convolutional network, especially efficient for image segmentation.
- **CycleGAN** [73]. A generative adversarial network, well-known for image-to-image translation.
- **Deep3D** [70]. A widely-used 2D to stereoscopic 3D image conversion approach using deep convolutional neural network (CNN).

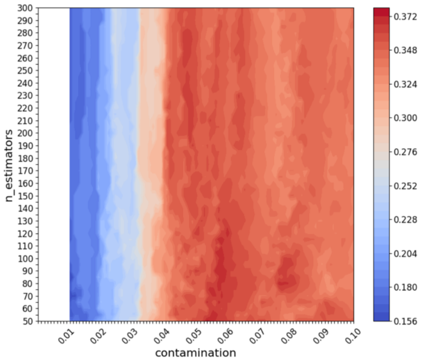


Fig. 5. Experiment results of hyperparameter settings of inconsistency detector (F-1 score)

- **DIBR** [38]. The state-of-the-art 2D to 3D image conversion approach, utilizes a deep CNN alongside a fast inpaint algorithm.

We also use a variant of our model without depth guidance as a baseline:

- **Painter without Depth Guidance (Painter$_{ND}$).** Another version of our Painter with depth guidance removed. The whole framework without depth guidance is referred to as StereoID$_{ND}$.

4.4.2 For SVI Issue Detection of StereoID. Use the stereo generation method in Section 4.4.1, our distance calculator, and our inconsistency detector for issue detection. The stereo generation methods include ResNet-Based Autoencoder, U-Net, CycleGAN, Deep3D, DIBR, and Painter$_{ND}$.

4.5 Evaluation Metrics

4.5.1 For Stereo Generation of Painter. We adopt three distinct evaluation metrics in experiments.

- **L1.** The mean absolute error between the synthesized right-eye image and the actual one.
- **L2.** The mean square error between the synthesized right-eye image and the actual right-eye image. Smaller L1 and L2 values indicate better performance in terms of pixel-level accuracy.
- **SSIM (Structural Similarity Index Measure).** Unlike traditional metrics like MSE (Mean Squared Error), SSIM considers changes in structural information, luminance, and contrast. It provides a more perceptual assessment, aiming to capture the human visual system's judgment of quality. Higher SSIM values indicate better performance of Painter.

4.5.2 For SVI Issue Detection of StereoID (RQ3). A comprehensive set of evaluation metrics was chosen to assess StereoID. These metrics enable both a granular insight into the detection of each class and an overall performance understanding. Specifically, we utilize:

- **Precision (-1)**: Quantifies the proportion of true SVI issues among the instances labeled as anomalies by the model. It is defined as:

$$P_{-1} = \frac{TP_{-1}}{TP_{-1} + FP_{-1}} \tag{11}$$

- **Recall (-1)**: Measures the proportion of actual SVI issues that were correctly detected.

$$R_{-1} = \frac{TP_{-1}}{TP_{-1} + FN_{-1}} \tag{12}$$

- **F1-score (-1)**: The harmonic mean of Precision and Recall for SVI issues, offering a balance between the two. It is computed as:

$$F1_{-1} = 2 \times \frac{P_{-1} \times R_{-1}}{P_{-1} + R_{-1}} \tag{13}$$

Table 2. Performance of depth-aware conditional stereo image translator (Painter) compared with baselines

Approach	Autoencoder		U-Net		CycleGAN		Deep3D		DIBR		**Painter$_{ND}$**		**Painter**	
Metrics	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std
L1 ↓	0.1737	0.1117	0.0521	0.0356	0.0230	0.0382	0.2966	0.2086	0.0381	0.0465	0.0219	0.0372	0.0204	0.0366
L2 ↓	0.2141	0.1138	0.0862	0.0456	0.0553	0.0575	0.3517	0.2069	0.0861	0.0709	0.0555	0.0556	0.0495	0.0529
SSIM ↑	0.3639	0.2534	0.5223	0.3013	0.8287	0.1667	0.2840	0.2409	0.7903	0.1957	0.8417	0.1622	0.8638	0.1455

- **Precision (1)**: Analogous to Precision (-1), but for the normal class.
- **Recall (1)**: Similar to Recall (-1), but captures the detection rate of the normal instances.
- **F1-score (1)**: Represents the balance between Precision and Recall for the normal instances.
- **Accuracy**: Highlights the proportion of all instances (both SVI issues and normal instances) that are correctly classified. It's given by:

$$\text{Accuracy} = \frac{TP_{-1} + TP_1}{TP_{-1} + TP_1 + FP_{-1} + FP_1 + FN_{-1} + FN_1} \tag{14}$$

- **Macro avg**: The average score across both classes, treating them equally irrespective of their imbalance.
- **Weighted avg**: The average score across both classes, but weighted by the number of instances in each class to account for potential class imbalances.

Where TP_{-1}, FP_{-1}, and FN_{-1} denote the true positives, false positives, and false negatives for the anomaly class (-1), respectively, and similarly for the non-anomalous class (1).

For RQ2, since we don't have ground truths for the test set, we can only calculate the detected SVI issues, undetected SVI issues and recall on the subset of bug report screenshot.

5 RESULTS AND ANALYSIS

5.1 Performance of Stereo Generation (RQ1)

To evaluate the effectiveness of our proposed depth-aware conditional stereo image translator (Painter) of StereoID in generating stereo images, we conducted extensive experiments comparing StereoID with five baseline approaches: ResNet-Based Autoencoder, U-Net, CycleGAN, Deep3D, and DIBR, along with a variant of our model without depth guidance (i.e., Painter$_{ND}$). The comparison focused on three key metrics: L1 loss, L2 loss, and the SSIM, where lower L1 and L2 losses indicate higher pixel-level accuracy, and higher SSIM scores signify better structural and perceptual quality of the generated images.

Table 2 presents the performance of Painter compared with the baseline models. It is evident from the results that Painter achieves the lowest average L1 and L2 losses, at 0.0204 and 0.0495, respectively, and the highest SSIM score, at 0.8638. These results signify that Painter is capable of generating right-eye images with superior pixel-level accuracy and structural consistency compared to the baseline methods. Notably, Painter outperforms the Painter$_{ND}$ variant, demonstrating the importance of depth guidance in enhancing the quality of stereo image generation.

The superior performance of Painter can be attributed to its depth-aware image translation mechanism, which incorporates depth information to accurately model the spatial shift between the left and right-eye images. This depth awareness enables Painter to better capture the perspective shifts and occlusions that occur in stereo imaging, resulting in more accurate and consistent right-eye images. The high SSIM score further indicates that Painter maintains the structural integrity of the scene, which is crucial for avoiding disorientation and cybersickness in VR apps.

In contrast, baseline methods such as Deep3D and DIBR, while effective in certain contexts, exhibit higher L1 and L2 losses, indicating less accurate pixel-level image generation. Their lower

Table 3. Performance of SVI issue detection in each manifestation category, Y denotes detected SVI issues, N denotes undetected SVI issues[5]

Category	Y	N	Total	Recall
Object Omission	14	2	16	87.5%
Lighting and Shadow Discrepancies	10	2	11	90.9%
Object Position Discrepancy	5	4	9	55.6%
Shader Absence	3	3	6	50.0%
Unilateral Object Rendering	4	1	5	80.0%
Particle and Visual Effect Variations	5	0	5	100.0%
Material or Texture Mismatch	4	1	5	80.0%
Monocular Blindness	5	0	5	100.0%
Post-Processing Inconsistency	2	3	5	40.0%
Object Warping	4	0	4	100.0%
Level of Detail Inconsistency	3	1	4	75.0%
View Misalignment	3	0	3	100.0%
Warped Views	0	1	1	0.0%
Partial Object Rendering	1	0	1	100.0%
Asymmetric Viewing Angles	0	0	0	-
Other	2	0	2	100.0%
Total	65	17	82	79.3%

SSIM scores also suggest that these methods may struggle to preserve the structural and perceptual qualities of the original scenes when generating stereo images.

The results from our evaluation underscore the effectiveness of Painter in addressing the challenge of generating accurate and consistent right-eye images for VR apps. By leveraging depth information, Painter significantly enhances the quality of stereo-mapping generation, offering a promising solution to mitigate SVI issues.

5.2 Performance of StereoID: Detecting User-Repoted SVI Issues (RQ2)

This section evaluates the capability of StereoID in accurately identifying user-reported SVI issues across various manifestation categories (see Section 4.3).

Table 3 summarizes the detection performance of StereoID across each SVI category, presenting the number of issues detected (Y), undetected (N), the total number of issues, and the recall rate. The overall recall rate of StereoID stands at 79.3%, indicating a high degree of effectiveness in identifying SVI issues across most categories. Notably, StereoID achieved a 100% recall rate in several challenging categories, including Particle and Visual Effect Variations, Monocular Blindness, Object Warping, and View Misalignment, showcasing its robust detection capabilities even for semantically-complex SVI issues. The comprehensive coverage across diverse manifestations of SVI issues demonstrates the adaptability and generalizability of StereoID.

The analysis also reveals specific areas where StereoID could be further optimized, offering directions for future research and development. Categories such as Post-Processing Inconsistency and Shader Absence reflected not as high recall rates as other issue categories, suggesting areas for further refinement. This indicates the challenge of identifying subtle or semantic-driven inconsistencies, where the spatial or contextual cues are less pronounced. These findings open research opportunities for integrating more nuanced detection mechanisms or leveraging additional contextual information to improve detection accuracy in these areas.

Our results confirm that StereoID effectively identifies a wide range of SVI issues in VR apps, marking a significant advancement in automated SVI detection tools. By achieving high recall rates across various manifestations, StereoID provides a valuable resource for developers and quality assurance teams to proactively address and rectify SVI issues, thereby enhancing the overall quality and safety of VR environments.

[5] Some samples have multiple manifestations and have been categorized in multiple classes.

Table 4. Performance of SVI issue detection compared with baselines, -1 represents SVI issues, 1 represents normal cases

Metrics	Autoencoder	U-Net	CycleGAN	Deep3D	DIBR	**StereoID$_{ND}$**	**StereoID**
Precision (-1)	5.19	32.03	31.60	2.16	17.75	34.63	38.53
Recall (-1)	5.04	31.09	30.67	2.10	17.23	33.61	37.39
F1-score (-1)	5.12	31.56	31.13	2.13	17.48	34.12	37.95
Precision (1)	93.99	95.64	95.61	93.80	94.76	95.80	96.04
Recall (1)	94.16	95.82	95.79	93.98	94.94	95.98	96.22
F1-score (1)	94.08	95.73	95.70	93.89	94.85	95.89	96.13
Accuracy	88.85	91.95	91.90	88.50	90.30	92.26	92.71
Macro avg	49.59	63.84	63.61	47.98	56.25	65.21	67.28
Weighted avg	88.69	91.84	91.79	88.34	90.17	92.20	92.61

5.3 Performance of Detecting Wild SVI Issues (RQ3)

In this section, we delve into the performance of StereoID in wild real-world VR scenarios. This assessment is crucial for understanding the practical applicability and robustness of StereoID when faced with diverse and complex (industrial-setting) real-world VR apps. We compare StereoID against five baselines: Autoencoder, U-Net, CycleGAN, Deep3D, and DIBR, along with a variant of our framework, StereoID$_{ND}$, which lacks depth guidance. Performance metrics include precision, recall, F1-score, and overall accuracy for detected SVI issues (-1) and normal cases (1).

Table 4 presents the detection performance of StereoID and baseline methods. Notably, StereoID demonstrates superior performance across all metrics, particularly in the detection of SVI issues, with a precision of 38.53%, recall of 37.39%, and F1-score of 37.95%. These results are highlighted in comparison to normal case detection, where StereoID also leads with precision, recall, and F1-score of 96.04%, 96.22%, and 96.13%, respectively. Overall, StereoID achieved an accuracy of 92.71%, outperforming all baselines and evidencing its effectiveness in real-world scenarios. StereoID's leading performance in detecting SVI issues underscores its capability to accurately identify inconsistencies in complex VR environments. The improvement over baselines, especially in detecting SVI issues, is significant, considering the challenges posed by real-world apps, such as diverse visual effects, dynamic content, and varying levels of detail. The depth-aware mechanism within StereoID is pivotal for this success, enabling it to discern subtle discrepancies that are often missed by other approaches. The superior recall and precision in normal case detection further illustrate StereoID's robustness, minimizing false positives and ensuring reliable identification of non-issues. This balance is crucial for practical apps, where over-reporting can burden developers with unnecessary reviews, and under-reporting can leave critical issues undetected.

The evaluation results in the wild reinforce the potential of StereoID as a valuable tool for developers and testers of VR apps. By offering high accuracy and robust performance across a spectrum of real-world scenarios, StereoID stands out as a promising solution for enhancing the quality assurance process in VR development. Moreover, the results highlight the importance of incorporating depth information in SVI detection tools, opening avenues for future research to explore more sophisticated depth-aware approaches for even greater accuracy.

5.4 Statistical Significance Analysis

We conducted Mann–Whitney U test to analyze whether there exist statistically significant differences between the results achieved by our approach and those of baselines. This non-parametric test is particularly apt for analyzing data that do not necessarily follow a normal distribution, making it a robust choice for our evaluation. Specifically, the test is conducted under the distance calculation results between the generated right-eye images and actual right-eye images.

As shown in Table 5, these results collectively demonstrate the statistical significance of our method's performance when compared with baselines. The consistently low (or near-zero) p-values

Table 5. Results of statistical significance analysis

Analyzed Between Our Approach and Which Baseline	Autoencoder	U-Net	CycleGAN	Deep3D	DIBR	StereoID$_{ND}$
P-Value	0.0000	0.0000	7.8358e-19	0.0000	4.0191e-235	1.0528e-05

across different comparisons unequivocally suggest that our approach is significantly better than baselines in terms of detecting SVI issues.

6 THREATS TO VALIDITY

Internal Validity. StereoID partially depends on deep learning models, specifically conditional GAN models. These models inherently possess a risk of overfitting, which could lead to high performance on our dataset but limited generalization to unseen data. To mitigate this threat, we employ a rigorous training protocol, involving validation sets and early stopping mechanisms.

External Validity. While our empirical study incorporates a wide range of VR apps, it is conceivable that certain types of apps are underrepresented. This implies that our findings and the resulting deep learning model may not apply universally to all VR apps. Furthermore, our research is grounded in the current state of VR technology. Future advancements in VR hardware and software may necessitate adaptations in our methodology.

Construct Validity. We identify SVI issues based on a discrepancy metric between synthetic and actual right-eye images. While this discrepancy measure is intuitive and has demonstrated utility in our work, it may not capture all possible manifestations of SVI issues. Certain types of glitches might lead to negligible discrepancy according to our measure, yet still induce cybersickness in users. Furthermore, both the empirical study and data collection of one of our datasets rely on the analysis of bug reports from online forums. The inherent subjectivity and potential inaccuracies of these reports present a risk to the fidelity of our empirical study findings and the collected dataset.

7 RELATED WORK

7.1 Studies on VR Applications

Several empirical studies have been conducted to explore and understand VR apps. Rodriguez and Wang [61] studied the growing trends, popular topics, and common file structures of open-source VR projects. Adams et al. [28] conducted interviews with VR developers and users to understand their concerns about security and privacy. Li et al. [47] performed an empirical study on web-based extended reality bugs to understand their symptoms, root causes, and uniqueness. Nusrat et al. [55] analyzed the optimization commits in open-source Unity-based VR projects to better understand VR performance optimization. Epp et al. [35] studied the characteristics of VR games on Steam and players' complaints about these VR games. Rzig et al. [63] conducted a quantitative and qualitative empirical study to analyze existing testing practices of open-source Unity-Based VR apps. Recently, Li et al. [46] conducted a large-scale empirical study to model the software quality of VR apps from users' perspectives. Based on the findings of VR software quality, they discussed insightful implications of VR quality assurance for both developers and researchers.

7.2 Virtual / Augmented Reality Testing

In the realm of Extended Reality (XR, including VR/AR/MR) testing, significant strides have been made to address the unique challenges posed by these technologies [36, 59, 63, 65, 67, 68]. For VR testing, Wang [67] proposed a white-box framework, VRTest, to automate the testing of VR scenes by extracting scene information and manipulating the user camera with predefined testing strategies. Further refining the approach to VR testing, Wang, Rafi, and Meng [68] developed VRGuide. VRGuide employs a computational geometry method, Cut Extension, to optimize camera routes for comprehensive object interaction coverage in VR scenes. Gil et al. [36] proposed Youkai, a framework tailored for Android-based VR apps. Youkai's capabilities in object detection, camera position adjustment, and support for six Degrees of Freedom scenarios offer promising preliminary results for VR unit testing. Souza, Nunes, and Dias [65] introduced VR-ReST, which facilitates the specification of requirements through a semi-formal language and generates test data from these specifications. As for AR testing, Rafi et al. [59] proposed PredART to predict human ratings of virtual object placements in AR scenarios and further detect placement issues. Using automatic screenshot sampling, crowd-sourcing, and a hybrid neural network for image regression, PredART achieves effective results in placement issue detection. Given the lack of user-side end-to-end dynamic XR testing infrastructure, recently, Li et al. [45] proposed an interaction simulation and automated black-box testing framework for spatial computing XR apps.

Our work, StereoID, uniquely contributes to the VR/AR testing domain by focusing on the detection of SVI issues without the need for labeled data or access to internal application code. Unlike the methods mentioned above, which primarily target general testing challenges in VR/AR environments, StereoID specifically addresses the nuanced issue of SVI through a novel unsupervised, black-box testing framework. Our depth-aware left-right-eye image translator, integral to StereoID, signifies a groundbreaking step towards understanding and rectifying SVI issues in VR apps, pushing the boundaries of automated VR testing further.

7.3 Automated GUI Test Oracle

Recently, in order to address the limitations of regular test oracles in detecting GUI glitches, researchers have started exploring automated GUI test oracles that employ deep learning (DL) techniques for identifying abnormal GUI states in mobile and web apps [32, 49, 50, 59, 72]. These studies typically model GUI glitch detection as a classification problem and use different methods to perform buggy-side training data augmentation, generating screen captures with GUI glitches by either modifying normal GUI screenshots or injecting bugs into the application code and capturing the resulting screens. These data augmentation methods supply sufficient training data for deep learning classifiers to effectively detect faults. Such approaches yield promising results in their corresponding scenarios, significantly improving the efficiency of GUI testing.

Liu et al. proposed OwlEye [49], a deep learning-based approach to detect UI display issues such as text overlap, blurred screens, and missing images. The technique utilizes visual information from GUI screenshots. Chen et al. introduced GLIB [32], a technique designed for graphically-rich apps like games. GLIB detects non-crashing bugs, such as graphical glitches, using a code-based data augmentation technique. Zhao et al. presented Seenomaly [72], a vision-based linting approach for GUI animations. Seenomaly uses an unsupervised, computer-vision-based adversarial autoencoder to group similar GUI animations, even when lacking sufficient labeled data for training. This approach aids in linting GUI animations against design-don't guidelines. Macklon et al. [50] developed a technique for automatically detecting visual bugs in HTML5 canvas games by leveraging an internal representation of objects on the canvas. The method decomposes snapshot images into object images and compares them to oracle assets using four similarity metrics.

7.4 2D-to-3D Image Conversion

There exist multiple works in the computer vision domain that are related to 2D-to-3D image conversion. Deep3D by Xie et al. [70] uses deep CNNs for the automatic conversion of 2D videos and images into stereoscopic 3D formats. Unlike traditional methods requiring ground truth depth maps, Deep3D leverages stereo pairs from 3D movies for end-to-end training, showcasing significant advancements in performance and human subject evaluations. Lee et al.'s [44] multi-scale DNN approach redefines view synthesis from a single reference view by integrating a spatial transformer module within a unified CNN framework. Chen, Yuan, and Bao's DenseNet3D [31] introduces a novel application of 3D densely connected convolutional networks for automatic 2D-to-3D video conversion, achieving improved results and speed by incorporating 3D convolution layers to grasp the spatiotemporal video characteristics. The Fused Network proposed by Zhu, Liu, and Wang [74] integrates unsupervised depth estimation with DIBR in an end-to-end framework, demonstrating the benefits of leveraging both contextual and geometric information for view synthesis. Shih et al.'s [64] context-aware layered depth inpainting technique for 3D photography innovatively combines color and depth structure synthesis in occluded regions, offering an improved method for novel view synthesis in everyday scenes. Recently, Hachaj's [38] adaptable 2D-to-3D conversion approach utilizes a deep convolutional neural network alongside a fast inpainting algorithm, emphasizing the adaptability and efficiency of converting 2D images to 3D.

Our work, StereoID, distinctively focuses on the detection of SVI issues in VR apps, a niche yet critical aspect of the 2D-to-3D conversion domain. Unlike the aforementioned studies primarily aimed at enhancing the conversion quality or efficiency, StereoID tackles the challenge of SVI issues detection through an innovative unsupervised black-box testing framework. The novel depth-aware left-right-eye image translator, proposed by us, can generate high-quality synthetic right-eye images for SVI issue detection. Results in Section 5 demonstrate that StereoID outperforms the state-of-the-art 2D-to-3D image conversion approaches DIBR [38] and Deep3D [70].

8 CONCLUSION

This paper presents StereoID, an unsupervised black-box testing framework, to detect stereoscopic visual inconsistencies (SVI issues) in VR apps. Our empirical analysis of 282 real-world SVI issue bug reports from 15 VR platforms reveals the complexity and diversity of SVI issues, which are challenging for existing pattern-based supervised GUI testing methods. StereoID leverages a depth-aware conditional stereo image translator, Painter, to generate synthetic right-eye images from left-eye images, framing SVI issue detection as an anomaly detection problem. We construct several datasets to verify the effectiveness and usefulness of StereoID, including a large-scale dataset of over 171K VR stereo image screenshots collected from 288 real-world VR apps. Extensive evaluations demonstrate that StereoID effectively identifies SVI issues in both user-reported SVI issues and wild VR scenarios, outperforming baselines in pixel-level accuracy, structural consistency, and SVI issue detection accuracy. We believe our research significantly advances the quality assurance of VR apps, enhancing user experience and safety. Our future work will focus on refining and enhancing the framework to further improve robustness and applicability.

9 DATA AVAILABILITY

We release our datasets and StereoID, at https://sites.google.com/view/stereoid.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their helpful feedback. The work described in this paper was supported by the National Natural Science Foundation of China (Grant No. 62372219), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund), the Natural Science Foundation of Guangdong Province (Project No. 2023A1515011959), the Shenzhen International Science and Technology Cooperation Project (No. GJHZ20220913143008015), and the Shenzhen-Hong Kong Joint Funding Project (No. SGDX20230116091246007).

REFERENCES

[1] 2021. FILM XR. https://vrfilmreview.ru/.

[2] 2021. VirtualSkill - Virtual Reality Training. https://virtualskill.com/.

[3] 2021. XR Games. https://www.xrgames.io/.

[4] 2022. Oculus App Lab. https://developer.oculus.com/blog/introducing-app-lab-a-new-way-to-distribute-oculus-quest-apps/.

[5] 2022. Oculus App Store. https://www.oculus.com/experiences/quest/.

[6] 2022. SideQuest. https://sidequestvr.com/.

[7] 2023. Epic Developer Community Forums. https://forums.unrealengine.com/.

[8] 2023. GitHub. https://github.com/.

[9] 2023. GitHub Repository of ValveSoftware/SteamVR-for-Linux. https://github.com/ValveSoftware/SteamVR-for-Linux.

[10] 2023. Introducing SteamVR Home Beta. https://steamcommunity.com/games/250820/announcements/detail/1256913672017157095.

[11] 2023. Meta Community Forums. https://communityforums.atmeta.com/.

[12] 2023. Post-processing and Full-screen Effects. https://docs.unity3d.com/Manual/PostProcessingOverview.html.

[13] 2023. Post-processing Effects. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@12.0/manual/post-processing-effect-list.html.

[14] 2023. Stack Overflow. https://stackoverflow.com/.

[15] 2023. Steam Community. https://steamcommunity.com/.

[16] 2023. Unity Discussions. https://discussions.unity.com/.

[17] 2023. Unity Forum. https://forum.unity.com/.

[18] 2023. Unity Forum: Right Eye Discrepancies. https://forum.unity.com/threads/right-eye-discrepancies-oculus-urp-obi-fluid-shader.1047632/.

[19] 2023. Unity Issue Tracker. https://issuetracker.unity3d.com/.

[20] 2023. Unreal Engine Forums: Eyes Sometimes Show Different LODs in VR. https://forums.unrealengine.com/t/eyes-sometimes-show-different-lods-in-vr/389839.

[21] 2023. Unreal Engine Forums: Vive Eyes displacements. https://forums.unrealengine.com/t/vive-eyes-displacements/101890.

[22] 2023. Unreal Engine Issues and Bug Tracker. https://issues.unrealengine.com/.

[23] 2023. Visual Effect Graph. https://docs.unity3d.com/2023.2/Documentation/Manual/VFXGraph.html.

[24] 2023. VIVE Forum. https://forum.htc.com/.

[25] 2023. VIVEPORT. https://www.viveport.com/.

[26] 2023. VR Content on Steam App Store. https://store.steampowered.com/search/?vrsupport=401.

[27] 2023. VR Playtesting Guide. https://developer.oculus.com/resources/playtest-guide/.

[28] Devon Adams, Alseny Bah, Catherine Barwulor, Nureli Musaby, Kadeem Pitkin, and Elissa M. Redmiles. 2018. Ethics Emerging: the Story of Privacy and Security Perceptions in Virtual Reality. In *SOUPS*. USENIX Association, 427–442.

[29] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. PMLR, 214–223.

[30] Eunhee Chang, Hyun Taek Kim, and Byounghyun Yoo. 2020. Virtual Reality Sickness: A Review of Causes and Measurements. *Int. J. Hum. Comput. Interact.* 36, 17 (2020), 1658–1682. https://doi.org/10.1080/10447318.2020.1778351

[31] Bei Chen, Jiabin Yuan, and Xiuping Bao. 2019. Automatic 2D-to-3D Video Conversion using 3D Densely Connected Convolutional Networks. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*. IEEE, 361–367. https://doi.org/10.1109/ICTAI.2019.00058

[32] Ke Chen, Yufei Li, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Wei Yang. 2021. GLIB: Towards Automated Test Oracle for Graphically-Rich Applications. In *ESEC/FSE*. ACM, 1093–1104. https://doi.org/10.1145/3468264.3468586

[33] John W Creswell and Cheryl N Poth. 2016. *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications.

[34] Noureddine Elmqaddem. 2019. Augmented Reality and Virtual Reality in Education. Myth or Reality? *Int. J. Emerg. Technol. Learn.* 14, 3 (2019), 234–242. https://doi.org/10.3991/ijet.v14i03.9289

[35] Rain Epp, Dayi Lin, and Cor-Paul Bezemer. 2021. An Empirical Study of Trends of Popular Virtual Reality Games and Their Complaints. *IEEE Trans. Games* 13, 3 (2021), 275–286. https://doi.org/10.1109/TG.2021.3057288

[36] Adriano M. Gil, Thiago S. Figueira, Elton Ribeiro, Afonso R. Costa, and Pablo Quiroga. 2020. Automated Test of VR Applications. In *HCI International 2020 - Late Breaking Posters - 22nd International Conference, HCII 2020, Copenhagen, Denmark, July 19-24, 2020, Proceedings, Part II (Communications in Computer and Information Science, Vol. 1294)*. Springer, 145–149. https://doi.org/10.1007/978-3-030-60703-6_18

[37] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. *Advances in neural information processing systems* 30 (2017).

[38] Tomasz Hachaj. 2023. Adaptable 2D to 3D Stereo Vision Image Conversion Based on a Deep Convolutional Neural Network and Fast Inpaint Algorithm. *Entropy* 25, 8 (2023), 1212. https://doi.org/10.3390/E25081212

[39] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1125–1134.

[40] Hae-Gon Jeon, Jaesik Park, Gyeongmin Choe, Jinsun Park, Yunsu Bok, Yu-Wing Tai, and In So Kweon. 2015. Accurate depth map estimation from a lenslet light field camera. In *CVPR*. IEEE Computer Society, 1547–1555. https://doi.org/10.1109/CVPR.2015.7298762

[41] Wee Sim Khor, Benjamin Baker, Kavit Amin, Adrian Chan, Ketan Patel, and Jason Wong. 2016. Augmented and virtual reality in surgery—the digital surgical environment: applications, limitations and legal pitfalls. *Annals of translational medicine* 4, 23 (2016).

[42] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[43] Joseph J. LaViola. 2000. A Discussion of Cybersickness in Virtual Environments. *ACM SIGCHI Bull.* 32, 1 (2000), 47–56. https://doi.org/10.1145/333329.333344

[44] Jiyoung Lee, Hyungjoo Jung, Youngjung Kim, and Kwanghoon Sohn. 2017. Automatic 2D-to-3D conversion using multi-scale deep neural network. In *2017 IEEE International Conference on Image Processing, ICIP 2017, Beijing, China, September 17-20, 2017*. IEEE, 730–734. https://doi.org/10.1109/ICIP.2017.8296377

[45] Shuqing Li, Binchang Li, Cuiyun Gao, and Michael R. Lyu. 2024. An Interaction Simulation and Automated Testing Framework for Spatial Computing Extended Reality Applications. (2024).

[46] Shuqing Li, Lili Wei, Yepang Liu, Cuiyun Gao, Shing-Chi Cheung, and Michael R. Lyu. 2023. Towards Modeling Software Quality of Virtual Reality Applications from Users' Perspectives. *CoRR* abs/2308.06783 (2023). https://doi.org/10.48550/ARXIV.2308.06783 arXiv:2308.06783

[47] Shuqing Li, Yechang Wu, Yi Liu, Dinghua Wang, Ming Wen, Yida Tao, Yulei Sui, and Yepang Liu. 2020. An Exploratory Study of Bugs in Extended Reality Applications on the Web. In *ISSRE*. IEEE, 172–183. https://doi.org/10.1109/ISSRE5003.2020.00025

[48] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*. IEEE, 413–422.

[49] Zhe Liu, Chunyang Chen, Junjie Wang, Yuekai Huang, Jun Hu, and Qing Wang. 2020. Owl Eyes: Spotting UI Display Issues via Visual Understanding. In *ASE*. IEEE, 398–409. https://doi.org/10.1145/3324884.3416547

[50] Finlay Macklon, Mohammad Reza Taesiri, Markos Viggiato, Stefan Antoszko, Natalia Romanova, Dale Paas, and Cor-Paul Bezemer. 2022. Automatically Detecting Visual Bugs in HTML5 Canvas Games. In *ASE*. ACM, 15:1–15:11. https://doi.org/10.1145/3551349.3556913

[51] Michael E McCauley and Thomas J Sharkey. 1992. Cybersickness: Perception of self-motion in virtual environments. *Presence: Teleoperators & Virtual Environments* 1, 3 (1992), 311–318.

[52] Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).

[53] Justin Munafo, Meg Diedrick, and Thomas A Stoffregen. 2017. The Virtual Reality Head-Mounted Display Oculus Rift Induces Motion Sickness and is Sexist in Its Effects. *Experimental brain research* 235 (2017), 889–901.

[54] Sarah Nichols and Harshada Patel. 2002. Health and Safety Implications of Virtual Reality: A Review of Empirical Evidence. *Applied Ergonomics* 33, 3 (2002), 251–271.

[55] Fariha Nusrat, Foyzul Hassan, Hao Zhong, and Xiaoyin Wang. 2021. How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects. In *ICSE*. IEEE, 473–485. https://doi.org/10.1109/ICSE43902.2021.00052

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[57] Yi-Hao Peng, Carolyn Yu, Shi-Hong Liu, Chung-Wei Wang, Paul Taele, Neng-Hao Yu, and Mike Y. Chen. 2020. WalkingVibe: Reducing Virtual Reality Sickness and Improving Realism while Walking in VR using Unobtrusive Head-mounted Vibrotactile Feedback. In *CHI*. ACM, 1–12. https://doi.org/10.1145/3313831.3376847

[58] Rebecca A Penn and Michael C Hout. 2018. Making Reality Virtual: How VR "Tricks" Your Brain. *Frontiers for Young Minds* 6 (2018).

[59] Tahmid Rafi, Xueling Zhang, and Xiaoyin Wang. 2022. PredART: Towards Automatic Oracle Prediction of Object Placements in Augmented Reality Testing. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 77:1–77:13. https://doi.org/10.1145/3551349.3561160

[60] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. 2020. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence* 44, 3 (2020), 1623–1637.

[61] Irving Rodriguez and Xiaoyin Wang. 2017. An Empirical Study of Open Source Virtual Reality Software Projects. In *ESEM*. IEEE Computer Society, 474–475. https://doi.org/10.1109/ESEM.2017.65

[62] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 9351)*, Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi (Eds.). Springer, 234–241. https://doi.org/10.1007/978-3-319-24574-4_28

[63] Dhia Elhaq Rzig, Nafees Iqbal, Isabella Attisano, Xue Qin, and Foyzul Hassan. 2023. Virtual Reality (VR) Automated Testing in the Wild: A Case Study on Unity-Based VR Applications. In *ISSTA*, René Just and Gordon Fraser (Eds.). ACM, 1269–1281. https://doi.org/10.1145/3597926.3598134

[64] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 2020. 3D Photography Using Context-Aware Layered Depth Inpainting. In *CVPR*. Computer Vision Foundation / IEEE, 8025–8035. https://doi.org/10.1109/CVPR42600.2020.00805

[65] Alinne Cristinne Corrêa Souza, Fátima L. S. Nunes, and Márcio Eduardo Delamaro. 2018. An Automated Functional Testing Approach for Virtual Reality Applications. *Softw. Test. Verification Reliab.* 28, 8 (2018). https://doi.org/10.1002/STVR.1690

[66] Statista. 2022. Report of Active Virtual Reality Users Worldwide. https://www.statista.com/statistics/426469/active-virtual-reality-users-worldwide/.

[67] Xiaoyin Wang. 2022. VRTest: An Extensible Framework for Automatic Testing of Virtual Reality Scenes. In *ICSE: Companion Proceedings*. ACM/IEEE, 232–236. https://doi.org/10.1145/3510454.3516870

[68] Xiaoyin Wang, Tahmid Rafi, and Na Meng. 2023. VRGuide: Efficient Testing of Virtual Reality Scenes via Dynamic Cut Coverage. In *ASE*. IEEE, 951–962. https://doi.org/10.1109/ASE56229.2023.00197

[69] Chathurika S Wickramasinghe, Daniel L Marino, and Milos Manic. 2021. ResNet autoencoders for unsupervised feature learning from high-dimensional data: Deep models resistant to performance degradation. *IEEE Access* 9 (2021), 40511–40520.

[70] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. 2016. Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks. In *ECCV (Lecture Notes in Computer Science, Vol. 9908)*. Springer, 842–857. https://doi.org/10.1007/978-3-319-46493-0_51

[71] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853* (2015).

[72] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Seenomaly: vision-based linting of GUI animation effects against design-don't guidelines. In *ICSE*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1286–1297. https://doi.org/10.1145/3377811.3380411

[73] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In *ICCV*. IEEE Computer Society, 2242–2251. https://doi.org/10.1109/ICCV.2017.244

[74] Mingtong Zhu, Xiangning Liu, and Ronggang Wang. 2020. Fused Network for View Synthesis. In *3rd IEEE Conference on Multimedia Information Processing and Retrieval, MIPR 2020, Shenzhen, China, August 6-8, 2020*. IEEE, 303–306. https://doi.org/10.1109/MIPR49039.2020.00069

Received 2023-09-28; accepted 2024-04-16