Camera-Geometry Interpenetration in Virtual Reality

Ben Burgh, Kyle Johnsen

Abstract

One of the most significant challenges facing virtual reality experiences is the virtual world being misaligned from the real-world environment. This makes problems arise if a user ventures too far outward - if the virtual world is larger, then the user is at risk of running into walls in the real world, and, conversely, if the real world is larger, the user is at risk of running through virtual walls. In this study, we examined the second case to figure out the best practice for handling such scenarios. After reviewing how commercial experiences handle these events, we implemented ten techniques. We then conducted a user study, grading each method on comfort, intuitiveness, and immersion. We present results that provide strong evidence for a blend of a few techniques, none of which are yet popularly employed by commercial games.

Index Terms: H.5.2 [Information Systems]: Information Interfaces and Presentation—User Interfaces; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

1 Introduction

Camera-Geometry Interpenetration (CGI) is the issue in virtual environments where the camera can pass through simulated objects and walls without collision. In desktop environments, this issue commonly occurs when controlling a ”3rd person” camera that is aimed at an object. As the camera is moved, it may clip through geometry. A solution to this problem, if needed, is to simply keep the camera from clipping through the use of collision detection. The problem also occurs in virtual reality (VR), in the case of a tracked head-mounted display. User may move their head along a trajectory in the real world that in the virtual world moves the camera through geometry. As large-area 6-degree-of freedom, walk-around virtual reality (VR) systems using head-mounted displays (HMD) have become commonly available and applied to commercial games, such as those for the HTC Vive and Oculus Rift systems, this issue is becoming more pronounced. Furthermore, the solution of preventing interpenetration by moving the camera is ill-advised, as this may result in discomfort and nausea for users. Thus, the default solution may be to allow the camera to clip through walls, particularly when the application does not suffer from it.

However, there are several important cases where CGI could lead to undesirable results. Virtual reality has a long history of being used for training purposes, in addition to gaming and psychology applications. CGI is an issue in all these cases. First, it may cause a break in presence, particularly if the user accidentally causes CGI (e.g. by backing up into a wall they didn’t realize was there). For psychological applications, this abrupt change may be very concerning. Second, for immersive gaming, CGI is a threat to fairness and general game design. If a user has the ability to hide behind geometry and not be shot or peer through walls that should be opaque they will likely do so to gain an advantage. Similarly, when used for training, a virtual experience should be as realistic as possible and not allow – deliberately or inadvertently – trainees to leave the virtual boundaries.

Addressing this issue is generally challenging. For example, one commonly implemented solution is to turn the HMD screen completely black when CGI occurs, so the user cannot see anything on the other side until they return. However, the user may struggle to find their way back. This may also be a jarring change from the virtual world, reducing presence. Another solution is to prevent the user from escaping at all, by moving the virtual world as the user approaches the wall. While less intuitive, something like this would always keep the user within the confines of the virtual walls. But this introduces another issue of preserving the virtual space. If a user keeps walking into a wall and it keeps moving back, they would eventually reach the edge of the tracking space. If the user had no means of repositioning themselves within their space (for example, through teleportation) then they would have no good way to reset the space and re-center themselves. Furthermore, some solutions may be highly application dependent. For example, a single-player experience might care more about presence than a competitive multiplayer game, which care more about fairness. Some applications may be fine with moving the virtual space, while others might absolutely require it to be preserved. Others may choose a metaphor that is consistent with the application (e.g. the user is a ”ghost”).

To add clarity to the issue of CGI in immersive virtual reality, we began evaluating various approaches for dealing with CGI in immersive virtual reality environments. In this paper, we report our results from reviewing techniques used by existing commercial games. We also discuss how to implement several common techniques, some of which are our own custom designs. Finally, we report results from a user study (N=40) where participants rated each technique along several dimensions and ranked their favorites.

2 Related Work

While a large body of work exists on the subject of detecting and handling arbitrary collisions in virtual reality, the issue of handling CGI is comparatively much less explored. The Oculus Rift ”Best Practices Guide” suggests that developers should address the possibility that the user may stick their head through walls, but provides little guidance for how to do so [8]. Previous work has shown that, for hand collisions, visual perception dominates proprioception, and you can trick users into believing their hands are in a different place [1]. However, this same technique may translate poorly to camera collisions users would be likely to immediately notice if their head were in the wrong place, as the head position and eye positions are rigidly bound. This makes CGI a uniquely challenging problem in immersive virtual reality – going past a wall could mean a user gets trapped behind a wall and gets confused, sometimes even requiring outside assistance to recover [7].

A similar issue appears with non–immersive 3D applications, such as Autodesk AutoCAD and the Unity 3D editor. Users who are not adept at 3D navigation could easily move their camera through an object when they are trying to get a closer look. One solution to this use case is to add a rewind button, which allows the user to reset the camera back to a position it was at earlier [3]. In traditional non-immersive games, there are three options which are most commonly used [5]: The Ghost method, where users just pass through walls freely, is represented by doing nothing. The Clunk method, where users cannot move if it would put them in a wall, is represented by our Incremental Push method. The Slip method, where users can move parallel to and away from the wall, is represented by our Continuous Push method. Of these, the Slip method is the most

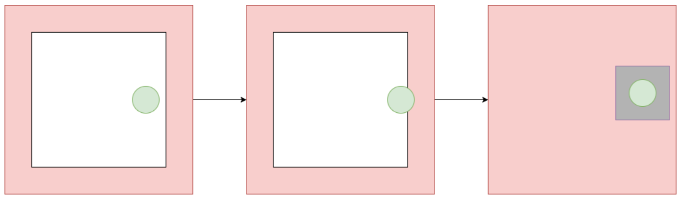

Figure 1: Disruptors – Once the user (Green) passes through the wall of the virtual room (White), the room disappears, and the user's experience suddenly changes.

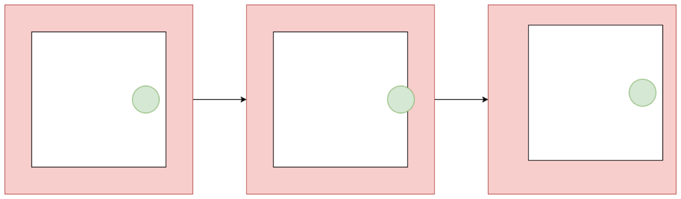

Figure 2: Anchors – Once the user passes through the virtual room, the virtual room moves so the user never leaves the room in the first place. However, the user is now closer to the edge of the real-world tracking space (Red).

widely used in non-immersive games.

A major issue with translating non-immersive methods to immersive experiences is that they may make users more likely to suffer from motion sickness, which is theorized to occur when head movement detected by the vestibular senses does not closely match head movement detected by optical senses [4]. Furthermore, attempting to persuade the user from not causing CGI is highly application dependent. For example, one approach is to use haptic feedback to warn the user of an impending CGI event [6]. Despite such solutions, no existing solution can fully prevent CGI through haptics.

Thus, for immersive experiences innovative solutions, such as those that have been presented for avoiding simulator sickness during joystick movement by lowering field of view [2], are needed that avoid vestibular-optical conflict. Moreover, generalizable solutions are desirable.

3 Techniques

Based on our evaluation of existing games, we implemented 10 generalizable solutions to the CGI problem. Here we describe how each was implemented, and discuss potential issues with the techniques.

In all cases, the same general method was used to detect CGI. An invisible collision volume (a sphere) was set to follow the tracked head position of the user each frame. If the capsule intersected any ”wall” geometry during the frame, it was stopped and returned to the nearest non-colliding point of contact. Whenever the displacement vector between the tracked head position and the capsule position became non-zero, the CGI response technique was activated. This requires that, even if the tracked head position returned to a valid position, the path to get there must have not been impeded. It also allows for handling cases where the user's head is penetrating a surface, but the eyes are not.

All solutions fall under the following three categories:

The first category, ”Disruptors” (See Fig. 1), change the camera rendering after passing through a wall. Disruptors are primarily concerned with maintaining alignment between the real and virtual space and not providing any new vantage point that would have otherwise been unreachable. They usually will use a return mechanism, such as an arrow, to instruct users how to return back to the virtual environment.

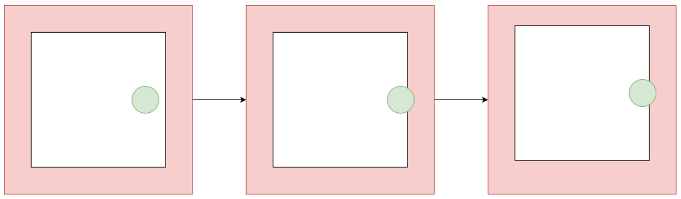

Figure 3: Preservers – The user is allowed to pass through the wall, and can see the virtual room from outside.

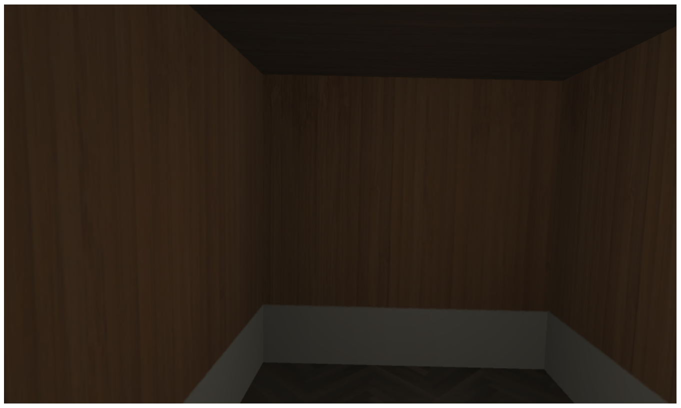

Figure 4: As a camera approaches the wall, its pixels darken. Once the camera is fully past the wall, it is completely black.

The second category, ”Anchors” (See Fig. 2), shift geometry (or move the user back) to prevent the camera from leaving a valid location. Unlike disruptors, the virtual environment is not modified. However, the alignment of the virtual environment with respect to the real environment may change.

The third category, ”Preservers”(See Fig. 3), have properties of both Anchors and Disruptors. Unlike Anchors, they allow camera to pass through the wall, but unlike Disruptors, they allow for some, but not necessarily all, otherwise unreachable viewpoints. They are primarily concerned with preserving vestibular-optical consistency.

3.1 Fade to Black

Fade to Black (Fig. 4) is the most commonly implemented disruptor technique. When the head is in collision the cameras begin to fade to black. As each camera passes through a boundary, it becomes completely black. This has the advantage of providing a warning, but never allowing the user to see beyond a wall. It also trivially handles arbitrary collision surfaces.

To implement Fade to Black, we apply a shader to linearly darken the image rendered to each eye's camera. A shader variable controls darkness by multiplying the RGB components of each rendered pixel by a value between zero and one, and a script changes this variable depending on the displacement of cameras during a CGI event.

3.2 Arrow Return Mechanism

One problem with Fade to Black is that users may not know how to return to a valid point. The Arrow (Fig. 5) technique attempts to address this problem with a 3D arrow which is always in front of the user's view, and always points towards the inside of the virtual room.

Arrow is implemented similarly to Fade to Black, but additionally a 3D arrow is rendered such that every frame it moves in front of the user and rotated to point back into the displaced head object.

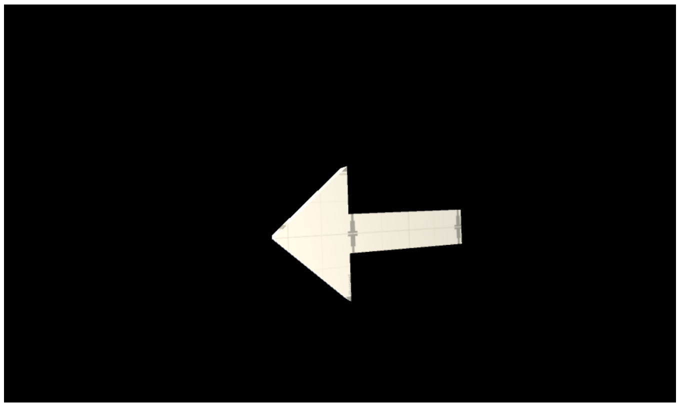
Figure 5: Arrow provides an intuitive way to get back into the virtual room

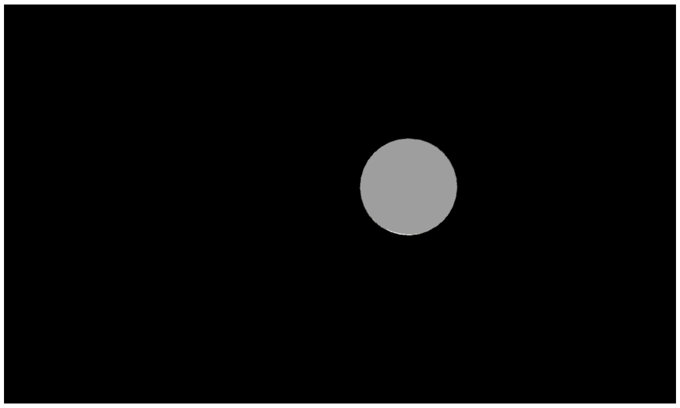
Figure 6: The ball shows the user where they were last in the virtual room

3.3 Ball Return Mechanism

Ball (Fig. 6) is another variant of Fade to Black. Once the camera becomes fully black, a Ball is enabled and shows the user's last valid position in the virtual space. Our implementation has the ball floating by itself, with everywhere else being black. This should provide less disruption than Arrow Return, but may be less intuitive. If the user is moved forward through a wall, it will only show black until the camera is turned around.

The implementation of Ball Return is similar to Arrow. Once the user passes the wall, the room disappears, leaving a black space. Additionally, a 3D sphere is enabled to represent the user's last position in the room.

3.4 Stop Rendering

The Stop technique is a severe disruptor. Once CGI occurs, the camera view is no longer updated. This means that, no matter where the user looks, they will see the same image they saw on the frame before they passed the wall. This might give the impression that the room is rotating with the user's head.

This technique is intentionally unintuitive and disorienting. The purpose of including this in the experiment was to have a clearly inferior technique to validate rating scales and provide a basis for comparison with other techniques.

Stop is simple to implement. When CGI occurs, the camera rendering is disabled. The result is that, until the user enters the virtual room again, they will see the same image.

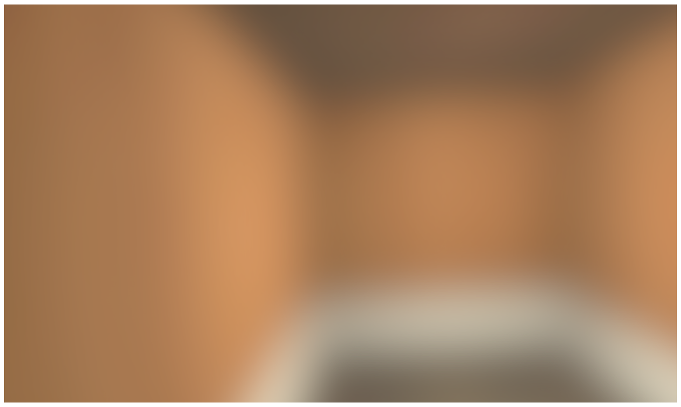
Figure 7: Blur lets users see enough to make out their position in the virtual world, but stops them from making out fine details.

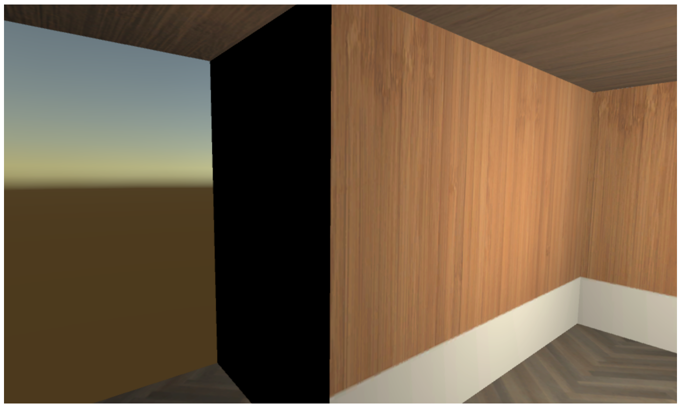
Figure 8: Ghost allows the camera to pass through the surfaces unimpeded. This can result in viewing through walls or outside the rendered environment.

3.5 Blur

Blur (Fig. 7) is a preserver and partial disruptor. When the camera passes through a wall, the view begins to blur. This provides immediate feedback, telling the user that something has happened, while still allowing them to see where they are in the virtual space, albeit not as clearly as before.

Blur is implemented using a full screen shader that activates during CGI. The shader performs a Gaussian blur on a down sampled version of the rendered image. An input to the shader controls the level of blur, such that the blur can be modified to be less severe at first CGI, gradually becoming blurrier as the displacement vector increases. However, it would be application specific to control the blur level appropriately. Thin walls would require more blur than thick walls to prevent the user from seeing beyond the wall.

3.6 Ghost

As discussed earlier, Ghost (Fig. 8), is both the most common and easiest to implement Preserver ”technique”. In fact, a simple implementation of ghost requires almost no effort at all on the part of the developer. Without a collision response, the virtual camera, when getting close to a surface will first clip wall geometry due to the near clipping plane. After passing through, while still within the wall, backface culling will often mean that the user is able to see back into the room. More complex implementations may fade away geometry as it is passed through.

A strength of this technique is that it does not impact vestibular-

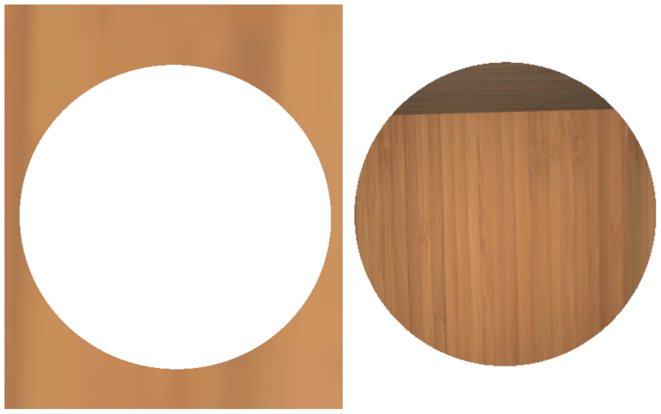

Figure 9: Left: A user approaching the wall sees a ”window” opening to a different place. Right: The user looks back through the window and sees the virtual room they came from

optical consistency. The user feels themselves physical move, and so movement through the virtual wall may be plausible. However, this technique has severe drawbacks if users are able to get to otherwise unreachable, undesirable locations by ghosting through walls or to see things that would otherwise not be visible (e.g. looking beyond a locked door).

3.7 Window

Window (Fig. 9) is a unique preserver technique inspired by the commercial game Dimensional (http://www.dimensionalgame.com/). As a user approaches a wall, a circle opens up on the wall, revealing what lies past the wall. The circle increases in size as the user gets closer to the wall, until it reaches the size of the user’s head at the point of intersection.

Once the user is through the wall completely, the circle (”Window”) stays in place at the same size. They are able to look back and see the original virtual room through the window, but they can also look around to see a new environment. This has elements of a disruptor, although it may be plausible that a white room exists beyond the virtual wall and an arbitrary room could be rendered instead of a white room, so it fits more clearly within the preserver category.

A custom shader was designed to implement Window. It used two variables to create the circular window – a fixed4, which represents a position in 3D space, as well as a float, which represented the size of the window. These two values defined the center and radius of a sphere. Any fragments rendering within the sphere are discarded. As such, the the user will see through the circular cross-section of the sphere. Every frame, the fixed4 value representing the position of the sphere is updated to the position of the user’s head as long as they remain inside the room.

There are a few weaknesses with this technique. First, the shape of the window might not always be the same. This could occur if the wall’s surface is curved, as the the window might not appear as a cross-section of a sphere. Or, if the user is approaching a corner, a window will open from both walls. Finally the technique is more complex to implement than others, as it requires the custom shader for every wall in the environment. Furthermore, it requires additional development time to stop objects beyond the wall from rendering, if desired, and in the case of complex walls, defining what is beyond a wall is challenging. One approach to do this could be to perform a raycast through the window, disabling geometry that is not hit by the cast, or to insert additional clipping planes just beyond the wall.

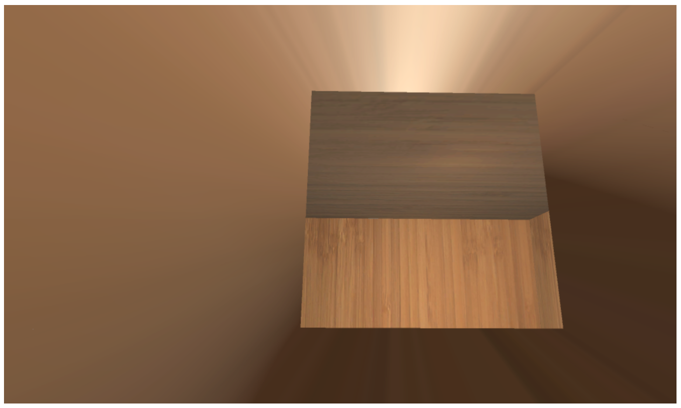

Figure 10: In Tunnel, the user morphs the shape of the wall

3.8 Tunnel/Rubber Wall

Tunnel (Fig. 10) is a preserver technique that we designed as a variant on Window that may be more plausible. During a CGI event, a tunnel encloses a volume from the user’s head to the point of intersection. The metaphor for this is that the walls are made of rubber and the user pushes into them This allows the user to turn around to see the room they entered, giving them a clear route back. Also, to enhance the illusion, the interior of the tunnel is rendered with the texture of the wall.

To implement tunnel, the users head is modeled as a sphere with a radius r. Once the sphere intersects a wall, the tunnel is created at the point of collision. Firstly, we generate a quad with a depth mask over the wall, creating the image of a hole in the wall. Then, we generate another quad of the same size, which is placed on the sphere at the furthest point into the wall. This quad is given the same texture and UV coordinates as the section of the wall that is hidden by the depth mask. Then, we create four more quads to connect both quads on all sides and create a cube. These four quads are textured appropriately so that they connect the original wall with the quad in the back. This makes the transition smoother. The position of the five interior quads is recalculated every frame to account for the users head movement.

This implementation currently has challenges dealing with arbitrary wall surfaces. First, it has no way to handle corners whichever surface the sphere hits first is the one it will stay on. Additionally, it has a similar issue with curved surfaces where it pokes out. Another issue is that, if the user moves too far parallel to the wall, the quads will converge, making it possible that the users eyes will actually be outside the tunnel, destroying the illusion. Our implementation sufficed for the study, but it may be difficult to generalize.

We also implemented an alternative to this technique that smoothly deforms all vertices of the wall mesh away from the displaced head. This approach is promising, and generalizable to many situations if the wall geometry has enough vertices for the appearance of a smooth deformation. This technique is currently quite computationally expensive for smooth effects, particularly when many walls are intersected simultaneously.

We chose to implement the earlier approach for the study, as this was thought to generate a superior effect, but recognizes that the challenges to its implementation warrant further consideration of the smooth vertex-deformation metaphor.

3.9 Incremental Push

Incremental push is an Anchor technique that moves the virtual room by a fixed amount once the user gets close to the wall, so that they never fully intersect in the first place. This is comparable to the player teleporting a short distance away from the wall. By doing so,

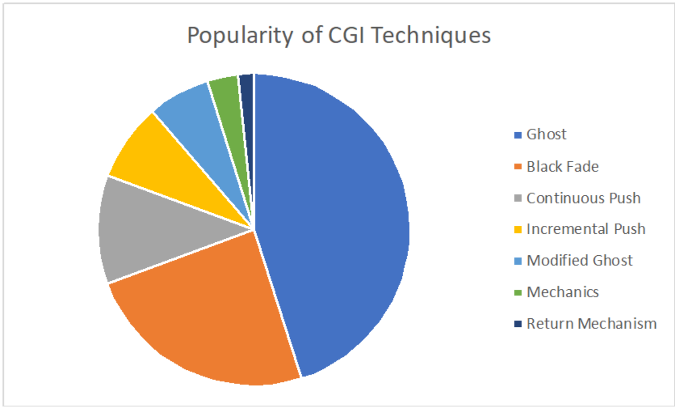

Figure 11: A breakdown of the most popular techniques to handle CGI used in commercial VR games and applications

the user remains in the VR space, instead of ever being out of bounds. However, this introduces the significant issue of space preservation. If the VR space is moved, then the danger arises that it could move too much. If the user is assuming the location of physical walls and objects in the real environment, this could lead to safety problems. This technique would be similarly incompatible with passive haptic techniques such as foam walls or tables in the environment.

In our implementation of Incremental Push, if the user gets too close to a wall, then the virtual room moves in the direction of the walls normal by a fixed distance. This handles corners fine, and works for walls of any size. It would have challenges with low virtual ceilings or raised platforms, as these would change the location of the physical floor relative to the virtual floor.

3.10 Continuous Push

Continuous Push is another Anchor technique that is partially space preserving. Rather than a fixed jump, the virtual space moves over only as much as the user pushes. It also moves back to the original space along the same deformation vector. This has two key differences from Incremental Push: The space only moves until theres no longer a collision, and the wall will follow the user until they return back into the space. In doing so, this method avoids the space preservation problem that incremental push has.

Continuous Push is implemented similarly to Incremental Push, but instead of a fixed distance, the virtual room moves only as far as the user pushes in. Additionally, to preserve the VR space, we keep track of the displacement, and the room moves back along this vector until it returns to zero in all directions. In our implementation, this occurs instantaneously, but could also be gradual at the risk of inducing significant vection.

4 Survey of Commercial Applications

Prior to starting this work and as motivation for it, we recognized that commercial VR games and applications had begun employing a variety of techniques to address CGI, many of which we found to be quite innovative. However, we were unable to find much literature describing the problem or potential solutions, suggesting that game designers were using ad hoc solutions. Thus, we performed a more comprehensive review, surveying 58 unique commercial VR applications to determine what techniques are the most popular. We conducted the tests by playing the game and attempting to stick our head through a virtual wall (if any) in the game and reported on what technique was used. Applications which did not have easily accessible walls were excluded from our tests. The overall breakdown of techniques by game is shown in Fig. 11.

Of the 58 tested, 28 used the ”Ghost” method, where users can freely move their head/camera past the wall and see what exists beyond the wall. In practice, this is likely popular because it is the default behavior of rendering algorithms if no solution is implemented. All other possible techniques require detecting collision and changing the experience. While we did not evaluate the potential impact of this ”solution”, we suspect that these games rely on the general preference of users to not stick their heads through walls. Other similar techniques were used. Two applications added blurring of the camera rendering after CGI. Another application changed the rendering to black and white.

The second most popular technique, which was used by 15 of the 58 applications, was ”fade or cut to black”. In this method, once a collision occurred, the screen either faded to black or cut to black instantly. This gave the user immediate feedback, but after becoming black, it was not inherently obvious for the user which way to return to the room. One application addressed this problem with an arrow that pointed towards the return trajectory. This provided an intuitive method telling the user where they should go in order to return back to non-CGI. Though more involved than ghosting, fading to black is relatively straightforward to implement with basic collision detection techniques and a custom camera shader.

The next most popular technique is an adaptation of the ”Slip” method, and was used by 7 applications in the survey. In this adaptation, if the user moves towards a virtual wall, their movement will be impeded – they will find themselves moving in the real world, but against the same wall in the virtual world. In other words, the component of camera movement in the direction towards the wall surface is removed.

The fourth most popular technique, which was used by 5 applications, is similar to the ”Clunk” method. If the user’s movement would put them past a wall, then the user’s avatar is translated a short distance in the direction of the wall’s normal vector.

Issues arise from the use of Slip and Chunk, because, when a user walks around in their real space, under normal circumstances their avatar in the virtual world undergoes the same movement. However, if an application uses Slip or Clunk, then this is not necessarily the case. As this constitutes a vestibular optical conflict, it may increase the likelihood of simulator sickness. In addition, it creates a problem of ”space preservation”. If a user is teleported away from the virtual wall, while moving towards the real-world tracking boundaries, then their new resting position will be closer to the tracking boundaries. Depending on the real-world environment, this could be a nuisance, if the user loses tracking, or dangerous, if the user runs into a real-world wall unintentionally.

The final 3 applications used game mechanics to disincentivize and even incentivize users from colliding with walls. One application caused the user’s avatar to die, causing a loss of progress, while the other application, less punishingly, drained the user avatar’s hit points resource. Lastly, one innovative application had a unique variant of the Ghost method, where objects close to the user’s head would ”dissolve” and reveal the contents inside. This technique was very interesting because it provided a more plausible visual explanation for CGI.

In total, 58 games were chosen from the Steam and Oculus stores. 14 were exclusive to the Oculus Rift, 32 exclusive to the Vive, and 12 were compatible with both. 23 of the applications were free-to-play, 13 of which used the Ghost method, which is roughly the same ratio. However, 12 of the applications were multiplayer, and only 4 used the ghost method. This shows that many developers are already wary of allowing users to have an unfair advantage by looking past walls.

5 User Study

5.1 Design

To evaluate the techniques, we conducted a user study involving 40 participants from the local student and faculty community of our University. We used a repeated-measures design, such that each participant tried every technique, each user tried them in a different order. As the number of distinct orderings of 10 levels is enormous, we used a 10 X 10 Latin square ordering rather than purely random order. This resulted in each order being assigned to exactly 4 participants.

The virtual environment used for the study was developed using Unity 3D and was purposely abstract and simple. It was modeled as an ordinary size room with four identical walls, as well as a floor and a ceiling. There were no doors, windows, or other openings. The HTC Vive (1080x1200 pixels per eye, 90Hz, 110° FoV) virtual reality headset and tracking system was used for the experiment.

Two instruments were designed for the study. The first was a background survey that participants filled out prior to beginning the main experimental procedure. Participants self-reported gender, age as well as ownership of phone-based and computer-based virtual reality systems and experience with gaming, as these aspects were hypothesized to be influential in determining the relative ranking of techniques. The second instrument was filled out by the experimenter. It included a 3 **ratings**, each on a scale from 0 (lowest) to 10 (highest), for the "comfort", "intuitiveness", and "immersion"(presence) of each technique (a total of 30 measurements for all techniques combined). Participants were also asked to specifically choose their **top 3 and bottom 3** techniques. Participants were also asked to **comment** on what they liked or disliked about each method. We believed it was important to distinguish which methods were scored higher, versus which methods were users' favorites. Additionally, we computed **rankings** for each method from participant ratings to better capture relative differences and reflect better reflect any consensus among participants. Ratings were sorted from highest to lowest and then assigned a corresponding ranking of 1 to 10. Ties at two ranking positions were averaged, such that the sum of all rankings was always 55 (e.g. a ranking of 1,1,3,4,...) would become (1.5,1.5,3,4...).

5.2 Procedure

The study procedure was non-standard due to the large number of methods evaluated and the need to compare methods to each other reliably. After filling out the background survey, they were asked to put on the stereoscopic headset. They then were asked to get comfortable moving around the virtual space. After becoming acquainted with the motion tracking, the experimenter had the participant evaluate each method. The order of first presentation was determined by the Latin Square, but participants could, at any time, request to revisit a prior method to make a comparison. In addition, prior to making any evaluations, participants were shown the first two methods in their list. We chose to start with two methods to give participants a better frame of reference for the different methods they would experience. After providing ratings for the first two, participants were shown and asked to rate methods one at a time. Once participants rated all methods, we asked them choose three favorite and least favorite methods as described above. Participants could adjust their ratings at any time and could ask for clarifications about the experimental variables at any time. This was done to ensure as accurate of a ranking as possible, limiting recall bias and confusion about the terms. Note that we used the word "immersion" as the term for the "feeling of being there", which was explained to participants, as it is more commonly known than the term "presence".

6 Results

All 40 participants (29M, 11F) tried all ten techniques, and provided all requested ratings. This resulted in 400 total observations with independent and dependent variables for participant ID, Method, Comfort, Intuitiveness, Top3, Bottom3 and computed variables for Average Rating, Comfort Ranking, Intuitiveness Ranking, Immersiveness Ranking, and Average Ranking. Data was primarily analyzed using IBM SPSS version 24.

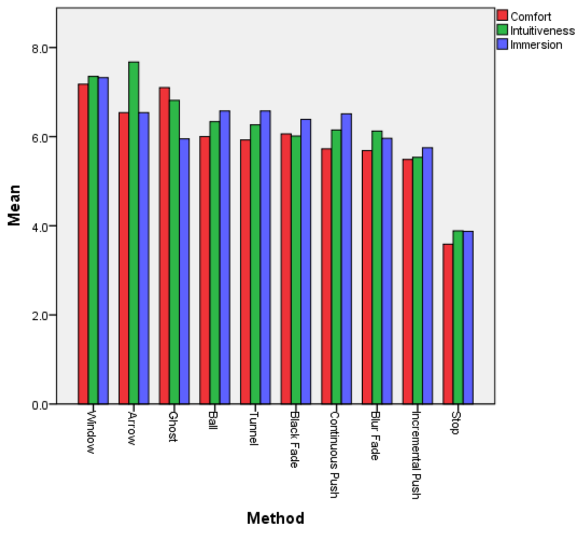

Figure 12: Ratings of Each of the three Category for each method, sorted by average

Method	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
Arrow	6.917	.345	6.219	7.615
Ball	6.304	.321	5.655	6.954
BlackFade	6.154	.327	5.493	6.816
BlurFade	5.925	.334	5.249	6.601
ContinuousPush	6.129	.357	5.406	6.852
Ghost	6.621	.247	6.120	7.121
IncrementalPush	5.592	.352	4.880	6.303
Stop	3.783	.401	2.973	4.594
Tunnel	6.254	.300	5.647	6.862
Window	7.283	.271	6.735	7.832

Table 1: Average ratings across 3 categories

Fig. 12 and Table 1 show category ratings for each technique. The average rating across all techniques was 6.10 (SD 2.53). The average rating given to techniques rated as a Top3 was 7.55 (SD 1.60), and the average rating for Bottom3 was 4.67 (SD 2.19). Participants reported the most favorable ratings for the Window, a preserver, across all categories (1st in Comfort and Immersion, 2nd in Intuitiveness). Arrow, a disruptor, was similarly rated highly across all three categories, especially Intuitiveness. The only method that received universally low ratings was Stop, as expected.

Fig. 13 shows how users categorized their favorite techniques. This was mostly consistent with average ratings. Again, Window was highly rated, being placed in the top 3 by 21 participants. Arrow was also highly rated by 19 participants. For bottom 3 groupings, 27 of 40 participants included stop, and 20 participant included incremental push. The most consensus (difference between top 3 and bottom 3 counts) was for Window (+19), Arrow (+11), and Ghost(+9) on the positive side and Incremental Push(-15) and Stop(-27) on the negative side. Others had little consensus (which could also be indifference) (difference <= 3). Others were all highly mixed, in particular Tunnel, which was in the top 3 for 12 participants, but the bottom 3 for 13 participants. Disruptors (except Arrow) all received

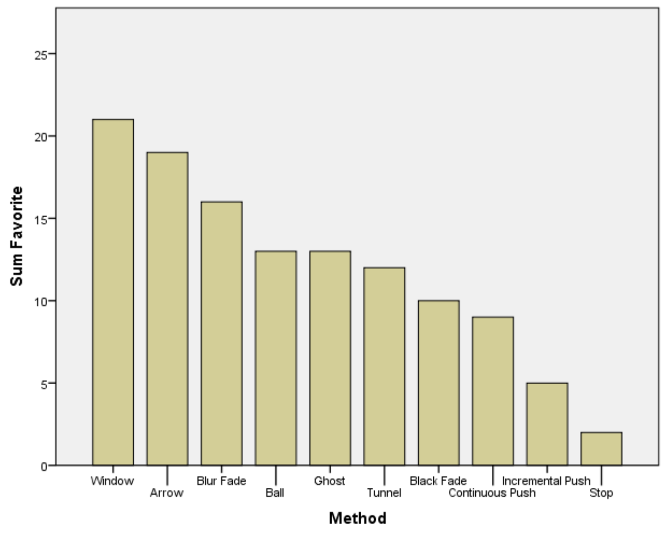

Figure 13: How many users reported each method to be their favorite (top 3).

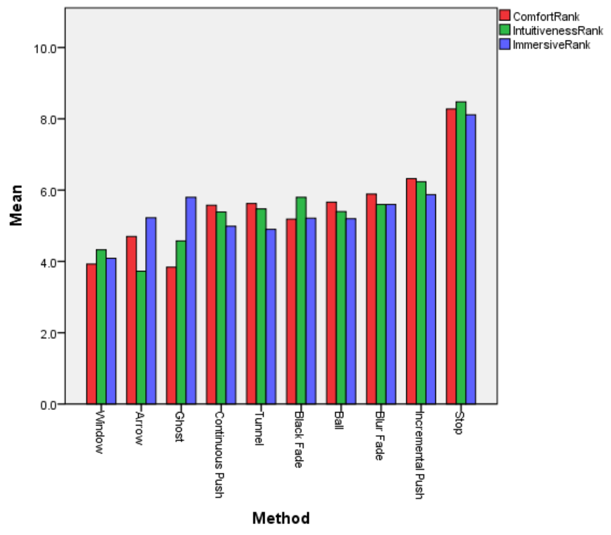

Figure 14: The average ranking of each method, calculated by user's ratings

10 votes in both categories as well.

Fig. 14 shows the computed average rankings for each method (inverted relative to previous graphs with 1 highest and 10 lowest). As with preference groupings, Window, Arrow, and Ghost were all highly rated. They differed primarily along the "Immersiveness" variable.

6.1 Analysis

The results suggested a strong preference among participants for Window and Arrow, with Ghost as an acceptable alternative. To determine the significance of these ratings, we conducted a repeated measures ANOVA, using method as the independent factor and average rating as the dependent factor. Results show a significant main effect of method (F(9,31)=12.95, Wilks' Lambda=0.21, $p<0.001$). Pairwise comparisons show many significant differences. Window was rated significantly higher than every method, other than Arrow. Arrow was rated significantly higher than Blur Fade ($p=0.02$), Incremental Push ($p=0.002$), and Stop ($p<0.001$). Stop was rated significantly lower than every other method ($p<0.001$ in every case). Incremental Push was better, but significantly lower than Ghost and Ball.

Further examining the distribution, we can gain a better understanding of how methods rank. As seen in Fig. 15, most of the techniques have a bimodal shape (2 humps). Arrow and Window have the highest probability of being liked. Stop clearly is disliked by a majority of users. Those with a bimodal shape may be highly application specific. Others with a flatter distribution may have less obvious application.

To qualify our results and analysis, we recorded and transcribed participants' conversations during the experiment. Of all participant's sentiments, there were six statements shared by at least three people.

First, 12 participants remarked that they "hated" Stop, because it was either "disorienting," "confusing," or just "weird." Several more people, while not using the word "hate", listed stop immediately after being asked which three methods were their least favorites (bottom 3).

Next, nine people reported that they initially did not like a particular technique, but after they knew what to expect they enjoyed it more. This was most often mentioned with regards to Incremental Push and Continuous Push, but two people said it in response to

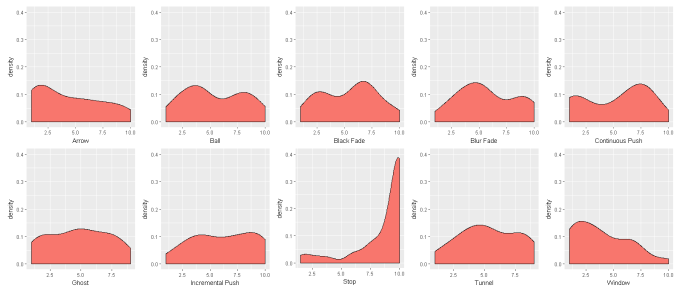

Figure 15: The rank distribution for each method. Techniques are listed in alphabetical order, from top-left to bottom-right

Tunnel or Ghost as well.

Eight people reported that they didn't like the sudden, jarring differences with Fade to Black and Ball. Interestingly, only one person reported the same issue with Arrow, which works almost identically. Three people said that they "loved" arrow because of how intuitive it was. They said that they immediately understood it. Providing immediate feedback may help users identify what is happening and quickly correct.

Five people said that they thought Window was "really cool." Additionally, one of the two people who listed Clunk in their top three said it was because it was cool. Coolness is a factor which is not represented by any of the three measures we recorded. However, coolness is certainly a factor that is important to video games, which represent a growing application category for VR.

Lastly, participant comments revealed that preference for Ghost may be dependent on what is shown outside of the virtual room. In our experiment the floor extended past the wall, with nothing of interest outside except the default skybox. Several participants reported that they liked Ghost because it was like walking out to a patio, but this was not intended. It's likely that users would be less receptive to Ghost if, for example, there was no floor outside the Virtual Room.

7 Discussion and Conclusions

Our analysis strongly supports that Window and Arrow are the most effective techniques. Window and Arrow both performed very well across all metrics. They were rated highly by the majority of participants, and chosen as favorites more often than any other technique. These methods are the best candidates for a preserver or disruptor technique.

Furthermore, while we rated them separately, several of the techniques are compatible with certain aspects of each other. For example, Arrow and Ball are additions to Black Fade. Given the popularity of Arrow, it should likely be the default over Black Fade. Window is highly compatible with Ghost. Ghost offers little advantage, and was rated significantly lower than Window. From our results, we can also conclude that Continuous Push is superior to Incremental Push, as it both preserves space and is preferred by users. Finally, we can conclude, with a high degree of certainty, that Stop should not be used, as we expected.

General guidelines for developers:

- If you want the most comfortable, or the most immersive experience, use Window.
- If you want an intuitive experience that is easy to implement and fair, use Arrow.
- If keeping the user in the same environment is a concern, use Continuous Push
- If CGI does not matter, use Ghost, but pay attention to what lies beyond the outer edges of the environment.

In addition, some techniques may fit better in different experiences. For instance, in a virtual maze, a user would easily be able to cheat using Ghost, so using some technique is almost necessary to preserve the maze's integrity. However, in a horror experience, designers could use CGI as a feature to include cryptic or scary images in ways that would not fit in other applications. Some applications may value certain features over others. For example, a single-player experience might value immersion highly, while a multiplayer application would prefer to avoid giving players an advantage. Ultimately, it becomes the job of the designers to choose how to handle edge cases like CGI, and we hope that our results provide some clarity on implementations to pursue.

8 Limitations and Future Work

There are several other possible techniques which we were unable to test to avoid fatiguing participants, or because of logistics. In particular, we did not test any game-mechanic-based disruptor techniques such as avatar death, since our environment was not a gaming application. Other variants we did not include are Continuous Push without preserving the space, instant black (versus fade to black), and a modified Ghost based on the circle Window with no secondary environment (in our case a white room). Our population may also not be predictive of the general population, as they were mostly Engineering students and faculty.

Additionally, we did not test in a more realistic environment. Our experiment environment was very sterile to keep users focused on the task and techniques rather than the experience. This empowered our study to find differences between the techniques. As a result, the relative value to an application's goals can not be determined from our study. It's also possible that contextualizing CGI would change user's perception on the different techniques handling them. This is especially true for a multiplayer experience, where users could potentially get an unfair advantage by peeking through a wall.

Furthermore, we did not evaluate how often users would stick their heads through virtual walls in the first place. We suspect that this is quite low, given results from other experiments that showed a high degree of collision avoidance in walk-around virtual reality [9]. The default solution, Ghost, may be the most comfortable and easiest for users.

To follow up on this study, we would like to test users' opinions of Ghost with different background environments (the "patio" comments we received). This could also apply to Window well, as providing a pleasing (or displeasing) view "inside" or beyond the walls is an intriguing concept.

References

[1] E. Burns, S. Razzaque, A. Panter, M. Whitton, M. McCallus, and F. Brooks Jr. The hand is slower than the eye: A quantitative exploration of visual dominance over proprioception. In *Virtual Reality, 2005. Proceedings. VR 2005. IEEE*, pp. 3–10. IEEE, 2005.

[2] A. S. Fernandes and S. K. Feiner. Combating vr sickness through subtle dynamic field-of-view modification. In *3D User Interfaces (3DUI), 2016 IEEE Symposium on*, pp. 201–210. IEEE, 2016.

[3] G. Fitzmaurice, J. Matejka, I. Mordatch, A. Khan, and G. Kurtenbach. Safe 3d navigation. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pp. 7–15. ACM, 2008.

[4] L. J. Hettinger and G. E. Riccio. Visually induced motion sickness in virtual environments. *Presence: Teleoperators & Virtual Environments*, 1(3):306–310, 1992.

[5] J. Jacobson and M. Lewis. An experimental comparison of three methods for collision handling in virtual environments. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 41, pp. 1273–1277. SAGE Publications Sage CA: Los Angeles, CA, 1997.

[6] R. W. Lindeman, R. Page, Y. Yanagida, and J. L. Sibert. Towards full-body haptic feedback: the design and deployment of a spatialized vibrotactile feedback system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 146–149. ACM, 2004.

[7] W. Stuerzlinger and C. A. Wingrave. *The value of constraints for 3D user interfaces*, pp. 203–223. Springer, 2011.

[8] R. Yao, T. Heath, A. Davies, T. Forsyth, N. Mitchell, and P. Hoberman. Oculus vr best practices guide. *Oculus VR*, 2014.

[9] C. A. Zanbaka, B. C. Lok, S. V. Babu, A. C. Ulinski, and L. F. Hodges. Comparison of path visualizations and cognitive measures relative to travel technique in a virtual environment. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):694–705, 2005.