VirtualSpace - Overloading Physical Space with Multiple Virtual Reality Users

Sebastian Marwecki[1], Maximilian Brehm[1], Lukas Wagner[1], Lung-Pan Cheng[1], Florian 'Floyd' Mueller[2], Patrick Baudisch[1]

[1]Hasso Plattner Institute
Potsdam, Germany

{surname.name}@hpi.uni-potsdam.de

[2]Exertion Games Lab, RMIT University
Melbourne, Australia

{floyd}@exertiongameslab.org

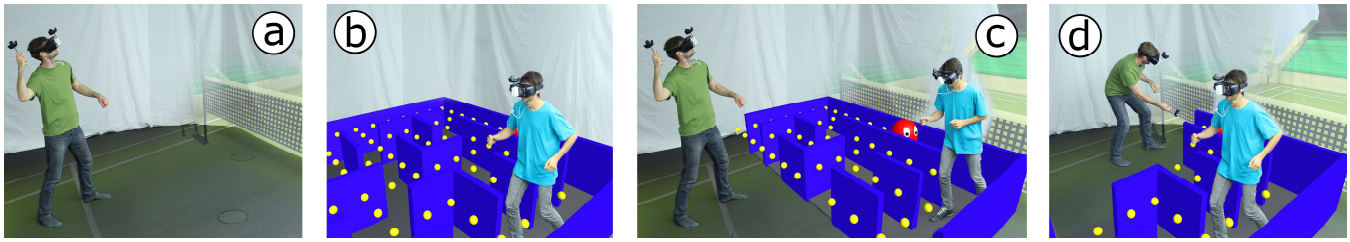

Figure 1: (a) This user is playing a badminton app. His side of the court fills the entire 4x4m tracking volume. (b) This other user is playing a Pac-Man game mapped to *the same* tracking volume. (c) VirtualSpace allows both users to share the same tracking space, without being aware of the other user. To keep users from running into each other, VirtualSpace limits each app to non-overlapping tiles at any given time. Client apps handle this in a way transparent to their users. The badminton app, for example, always makes the user's virtual opponent return the ball to locations inside the tile currently assigned to the app. (d) By reassigning tiles frequently, VirtualSpace moves users across the entire space, thereby seemingly allowing for unrestricted walking.

ABSTRACT

Although virtual reality hardware is now widely available, the uptake of *real walking* is hindered by the fact that it requires often impractically large amounts of physical space. To address this, we present VirtualSpace, a novel system that allows overloading multiple users immersed in different VR experiences into the same physical space. VirtualSpace accomplishes this by containing each user in a subset of the physical space at all times, which we call *tiles*; app-invoked *maneuvers* then shuffle tiles and users across the entire physical space. This allows apps to move their users to where their narrative requires them to be while hiding from users that they are confined to a tile. We show how this enables VirtualSpace to pack four users into 16m^2. In our study we found that VirtualSpace allowed participants to use more space and to feel less confined than in a control condition with static, pre-allocated space.

Author Keywords

Virtual reality; real walking; locomotion.

ACM Classification Keywords

H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

CHI 2018, April 21–26, 2018, Montréal, QC, Canada.

https://doi.org/10.1145/3173574.3173815

INTRODUCTION

Virtual reality (VR) offers a deep level of immersion if users are allowed to navigate by walking around in the physical world. This has been referred to as *real walking*. Compared to *simulated walking* using treadmills [7] and compared to locomotion techniques such as *teleportation* [6] or *walking in place* [24], real walking leads to higher levels of immersion [25].

With virtual reality headsets available to consumers [30] that are capable of tracking real walking in a room-sized tracking volume, one would expect real walking to become the dominant approach to VR. However, looking at experiences available for room-scale VR (e.g. Steam [23]), only a negligible percentage appear to be using real walking. The other experiences employ locomotion techniques, such as instant teleportation [6], despite the reduced experience. How can this be?

We argue that it is the physical space requirements of real walking that make it impractical for consumers. For example, in our city the rent of 4x4m space surpasses the cost of a VR headset and tracking system in a mere three months.

To reduce the space requirements of real walking, researchers have proposed several techniques. *Redirected walking* [22] folds long walking paths into a finite tracking volume, but requires very large installations (4x10m [22]). For room-scale installations, different approaches such as [21, 18] demonstrate how to reshape virtual experiences to fit arbitrary room shapes, which makes it easier to fit a real walking experience into an existing environment. This

approach is limited to applications, that can adapt to room setups and do not have specific space requirements.

In this paper, we propose a different technique, which is to *overload* multiple users into the same physical space.

VIRTUALSPACE

VirtualSpace is a novel system that allows multiple real walking VR users to share the same physical space without being aware of each other. *VirtualSpace* is designed to give each user the illusion of being in possession of the entire physical space.

Figure 1 shows an example. Two users share the same 4x4m physical space, while being tracked using a VR tracking system (Vive [30]). Both users are in their own, separate virtual environments. (a) The green user is immersed in a badminton app, while (b) the blue user experiences a Pac-Man game.

The key point is that *both* apps are mapped to the *entire* physical space, i.e., VirtualSpace allows each app to be designed under the assumption that the user has physical access to the entire 4x4m space. And that is true, albeit not necessarily at every particular moment, as VirtualSpace limits each app to a different non-overlapping tile of space. Client apps handle this in a way transparent to their user. The badminton app, for example, makes the user's virtual opponent return the ball always to locations inside the tile currently assigned to this app.

Managing space using "maneuvers"

To allow users to still complete their narrative and to prevent users from noticing that the system is confining them, VirtualSpace employs what we call *maneuvers*.

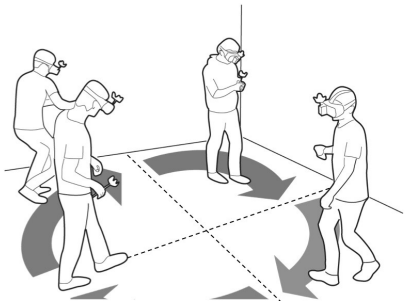

Figure 2: Users are confined into *tiles*. The *rotation maneuver* allows apps to move their user to the adjacent tile. This way users do not feel confined.

Figure 2 shows an example, here with four users in the same 4x4m tracking space, which is the configuration we used in our user study. When the user plays Pac-Man, the app needs to progress towards the area with the remaining pellets, but the segmentation into tiles prevents this progression. The player's app thus requests access to the desired tile, here the tile that is adjacent in clockwise order. As shown in Figure 2, VirtualSpace can provide the Pac-Man app with this access by rotating the entire field of all four users in clockwise direction. We call this the *(clockwise) rotation maneuver*.

Every maneuver comes with a certain start-up delay that allows all apps to get their users ready and every maneuver takes place at a certain movement speed. In this example, the apps may agree on a three-second delay and a one second transition speed.

As shown in Figure 3a, the Pac-Man app prepares for the maneuver by offering a virtual reward (the red cherry), yet it prevents the user from getting there by blocking the path using a ghost. Meanwhile the badminton app plays a slow ball to get the player synchronized with the timing of the upcoming maneuver.

Now the maneuver starts and every app moves their user to the new target position, here the next tile clockwise. The badminton app serves the user a stop ball that brings the player forward towards the net. As shown in Figure 3b, the Pac-Man app sends a ghost that chases the player down the corridor.

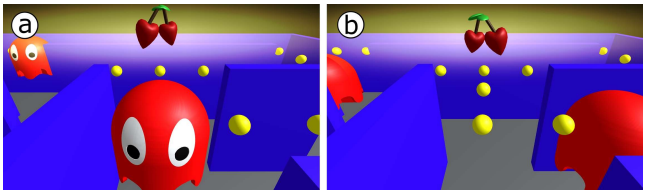

Figure 3: (a) The Pac-Man app guides users to follow the clockwise rotation maneuver by offering a virtual reward, here a cherry. (b) It prevents users' movement by placing ghosts in their way.

"Focus" and "Switch" maneuvers for quick actions

Clockwise and counterclockwise revolutions are sufficient in that it never takes more than two revolutions to send a user to any tile. Revolutions take time though, which is not always compatible with game action sequences that require fast, large, and erratic movements of a user. To enable such movement sequences, VirtualSpace offers the *focus* maneuver as shown in Figure 4. This maneuver allows a single app to temporarily take over most of the tracking space. Figure 4 shows an example when the badminton app uses a Focus maneuver to allow its user to hit the birdie in the center of the court in a quick succession of ball exchanges.

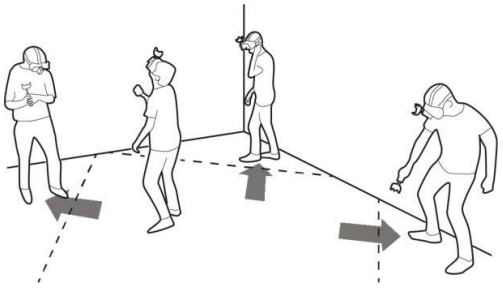

Figure 4: The *focus maneuver* temporarily provides the badminton app with control over most of the physical space. The other apps go into a *defocus state*.

While the one app has the focus all other apps "park" their users in a small amount of space at the rim of the tracking volume by providing them with a stationary task, e.g., by trapping the user in a corner (Pac-Man), or moles appearing in the bushes (Whac-A-Mole).

Focus maneuvers obviously take place at the expense of all other apps. Such "selfish" maneuvers are possible when all other apps agree to it, as will be detailed in the "Implementation" section. We also added a simple economic model in which apps can attach an added value (credits, money) to their maneuver request.

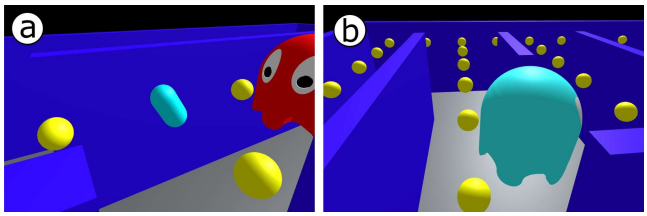

Figure 5: VirtualSpace allows for *in-app purchases*. Here the Pac-Man user can collect a blue pill which lets VirtualSpace value that app higher. That allows for a focus maneuver, which can be used to chase the ghosts.

Finally, the system allows for two users switching their rotational positions and thus for permutation of the rotation order. This enables certain users to move freely around the tracking space (e.g., Pac-Man) while others can stay at specific preferred areas (e.g., badminton).

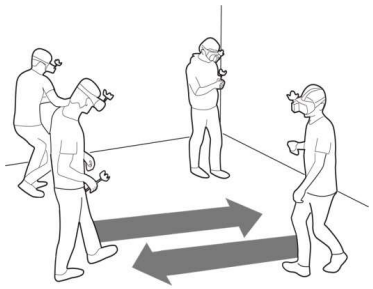

Figure 6: The *switch maneuver* allows two apps to switch tiles.

Supporting users joining or leaving in real-time

VirtualSpace is designed to run continuously with users joining or leaving at any time, any reasonable number of users and any combination of apps. Additional users can join as long as there is enough free space and leave at any time, thereby freeing up space. VirtualSpace continues to offer the same maneuvers irrespective of the number of users and the assigned tile size remains the same. However, if free space is available, apps can move on to empty areas, thus they do not need to perform the same maneuver. As shown in Figure 7, an app can request a maneuver, e.g. a rotation, without having another user move.

Figure 7: (a) VirtualSpace handles configurations with fewer users. (b) If free space is available, apps can perform different maneuvers at the same time, until (c) all space is allocated, then maneuvers need syncing.

Fallback

The assumption is that the apps succeed at confining their users to their tiles and at guiding their users to the agreed-upon target tiles. As with any real walking system this may fail, e.g., when users ignore the system and simply "wander off". VirtualSpace handles this situation like most real walking systems (such as Vive [30], or [17]) by (1) rendering a cage around the currently acceptable tracking space, i.e., the tile, before the user might exit and if the user does, (2) draws the outline of the users, so they can see each other and stop.

CONTRIBUTION

VirtualSpace's main contribution is a technique that allows multiple real walking virtual reality users to be overloaded into the same physical space. VirtualSpace achieves this by assigning each app to a smaller tile, where the overall tracking space is divided into computationally determined individual tiles. Frequent "maneuvers" allow apps to incentivize users to walk across the entire physical space, thereby allowing each app to progress its narrative and to prevent users from noticing that they are confined to a tile. This strategy enables VirtualSpace to achieve packings of high density, such as 4 users in 16m^2, as we demonstrate in our demo application (see "User Study" section).

Apps achieve the ability to run in VirtualSpace by implementing the VirtualSpace API (see "Implementation"), which potentially any external app could use (see "Application requirements"). We demonstrate how we do this by showing four examples (see "Applications").

VirtualSpace's main limitation is that even though apps can move users anywhere within the tracking space, getting there may be subject to a delay. Along the same lines, apps have to be able to generally comply when another app requests a maneuver. We found this to be more acceptable for apps with short interaction cycles, such as casual games or sports games rather than story-driven games, such as adventure games. We imagine, although not tested yet, that VirtualSpace can run one story-driven apps with any number of casual games.

Our current system and apps are limited to a certain number of users and a fixed tracking volume. Conceptually however the system can be extended to more users, multi-user applications, and spaces of different shape or size.

RELATED WORK

VirtualSpace builds on related work on single user locomotion in VR, real walking, shared tracking spaces, and overloading.

Single user locomotion and real walking in VR

While walking in VR can be enabled by treadmills [7], it is largely simulated with techniques as *walking in place* [24] or *teleportation* [6]. However, researchers agree that real walking, a continuous one-to-one mapping of physical to virtual locomotion, leads to the highest user satisfaction [25]. Real walking has high space requirements, which researchers have tried to reduce with several techniques, such as *resetting* [31], which rotates or repositions users once they hit the tracking volume's borders, or *seven league boots* [11], which virtually scale the user's movements. However, as these techniques perceptibly interrupt or alter the one-to-one mapping of physical to virtual movement, the immersive quality of real walking is reduced. The following techniques reduce space requirement for real walking, by altering the one-to-one mapping in a way imperceptible to the user, thus maintaining immersion. Razzaque et al. [22] found that redirected walking, which folds long walking paths into a finite tracking space, can lower the amount of space required for real walking, but as discussed earlier, redirected walking was found inapplicable for spaces smaller than 4x10m, without falling back to complementary techniques such as resetting. Vasylevska et al. [28] dynamically rearranged rooms in a virtual environment (aka *dynamic layout generation*) so that they unnoticeably overlap. Again, this requires a lot of space (9x9m). Both techniques succeed in space reduction insofar as they manage to virtually simulate endless space within a large physical space. VirtualSpace applies to any size of tracking space and reduces individual space requirements by letting users share the same tracking space.

Sharing tracking space

Real walking extends naturally to multiple users. Sra et al. [19, 20] developed VR applications that allow sharing the same physical space between multiple users. Of course, multi-user applications implicitly use the concept of overloading. However, we explore the wider concept of overloading space with users in *different* virtual environments, who are unaware of one another. The same authors, Sra and colleagues [21], adapted virtual environments to fit rooms of arbitrary shape, thus conceptually widened the possibilities for space setups also to mobile scenarios. This concept has been reused [18]. However, creating adaptive virtual environments might be difficult or sometimes not possible, as virtual experiences might have specific space requirements. Also, within mobile scenarios, the physical environment might need to be shared with others. Redirected walking has been shown to also apply to multiple users [1, 10], but still has a relatively high risk of collisions.

Overloading in non-VR application areas

The concept of overloading has been thoroughly explored. In computation alone, many parallels can be drawn, e.g. to management of memory space. We borrowed directly from those concepts: VirtualSpace (virtual RAM) has users in different environments (independent processes) quickly exchanging (dynamically loading) tessellated (fragmented) non-sharable tiles (partitions), allowing maneuvers like "switching" (swapping) or "de-focusing" (secondary memory). After we explored different memory schedulers (e.g., first-come-first-served, round-robin, shortest remaining time, and manual scheduling), our allocation algorithm basically uses fixed priority pre-emptive scheduling (described in the next section). Memory management is of course a faster process that takes place on a far smaller scale than our space management. Scaling up time and space to the other end of the spectrum allows comparisons to different fields, such as UIs [3], planning relocations of resources on construction sites [33] or reducing office space requirements for mobile workers (*open offices* [26, 27]), which underlines VirtualSpace's potential economic importance. With VirtualSpace we merely re-use existing overloading concepts and apply them to a multi-user VR setup to enable real walking. The main difference is that our system has users comply by using visual guidance.

Visual guidance based on incentives

VirtualSpace navigates users' through virtual environments by using virtual rewards and obstacles, here called incentives. This concept has been explored in many variations, e.g., by Peck et al. [15], who used "distractors" to navigate users in VR, but maybe most comparably by Fajen and Warren [8] who let users walk in virtual environments populated with goals and obstacles. They found that the attraction of the goal increased linearly with its angle from the current heading and decreased exponentially with distance, whereas the repulsion of the obstacle decreased exponentially with angle and with distance. We incorporated those findings in the placement of incentives in our apps (e.g., having both negative and positive incentives appear in the user's field of view and within close distance).

Unlike previous works, VirtualSpace allows multiple users in *different* virtual environments to share the *same* tracking space. It achieves this by borrowing from existing overloading strategies used for memory management. This leads to unprecedented reduction of space requirements. It is compatible with any app implementing our simple API.

VIRTUALSPACE API

VirtualSpace runs with any app implementing the VirtualSpace API, which acts in the following five steps.

VirtualSpace places and orients apps

When an app registers with VirtualSpace, the system arranges its place in the tracking volume, coordinating it with the other apps that are already running. Figure 8 shows an example. Here, the badminton app predicts an uneven spatial probability distribution for its user, as users tend to go back to the baseline whenever they can. Let us assume now that we have for example a second badminton user (playing against a separate AI). If two badminton apps were placed

in identical orientation, it would result in frequent collisions. VirtualSpace avoids this by requesting a sample of each app's spatial probability distribution. Then, the system tries out all possible positions and orientations of the apps to minimize overlap. In this example, the system decided to rotate the second badminton app by 180 degrees.

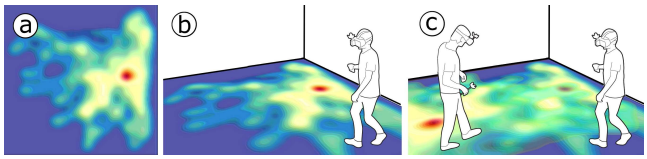

Figure 8: (a) Spatial probability distribution based on 3min of using the badminton app (axis lengths: 4m). (b) VirtualSpace places the first badminton user without rotational offset. (c) The second badminton user is placed at a 180-degree angle, minimizing the overlap in their probability distributions.

Apps inform VirtualSpace which maneuvers they favor

To help VirtualSpace perform maneuvers fitting all apps, each app informs VirtualSpace about the usefulness for each potential maneuver. The system deduces the potential maneuvers using a simple state machine (e.g., after the *focus maneuver* the system is in the "focus" state, no *rotation maneuver* is possible, but a *defocus* or *switch* maneuver). Apps then valuate potential maneuvers. The badminton app, for example, generally values those maneuvers highly that allow bringing the user back to baseline or to previously unused areas. Similarly, Pac-Man's erratic movements or badminton's reach typically require more space, leading to higher focus valuations. Additionally, apps provide the system with information on how fast they can comply with the suggested maneuver. They provide the preparation time (delay until the maneuver starts, in which incentive is placed) and the execution time (time for moving tiles, in which users follow incentives). This information can also be provided in linear dependencies (e.g., preparation and execution time add to a specific value).

VirtualSpace decides which maneuver to perform

The system now decides which maneuver to invoke. It adds up the utility for each maneuver reported by each app and picks the maneuver that maximizes utility across apps. A *focus* maneuver for the first app, for example, implies three corresponding *defocus* maneuvers for the other users, which might not have the same utility as a *rotation*. The system also tries to ensure that apps receive similar utility over time and tries to avoid maneuvers subject to timing mismatches between apps.

VirtualSpace synchronizes apps to start maneuver

When the system decides on the maneuver to perform, it also needs to determine the time until the maneuver starts. In badminton for example, this should be within the narrow time frame when the enemy AI can hit the birdie and the app can show the trajectory to the impact point. To ensure this, the system synchronizes the apps by using what we call *ticks*, events in which apps have the possibility to influence their user's movement. In the example of badminton, a tick is the moment the enemy AI hits. The moment that the system starts a maneuver should be when application ticks are in sync. The applications constantly provide information on their ticks using linear conditions, in badminton for example ticks occur once every full exchange. Each tick is given an allowed variance (which for our apps increases over time) together with a preference (e.g., badminton prefers quick exchanges but can let the AI play differently to synchronize). Ticks are synchronized by the system sending back master ticks, which the apps adapt to. The system can then compute the delay time, whose requirements were sent with the maneuver evaluations (see above). When a synced tick is then the same as the delay time, VirtualSpace can start the maneuver as we now ensured that every app can direct its users at that time.

VirtualSpace informs the apps, apps follow maneuver

The system now informs apps about the upcoming maneuver. It does so by sending a sequence of what we call *frames*, information about assigned areas at given times, from which apps can derive the need to place incentives. The system computes assigned areas, the tiles, as Voronoi tessellations to keep them convex, this enables apps to easily compute which paths their user can walk on. Apps respond to maneuvers by placing rewards and obstacles inside their virtual environment, thus guiding the user to the next tile (more detail in the "Applications" section).

The whole process described in this section is executed multiple times in parallel, to ensure quick interaction rates. While apps synchronize for the next maneuver, they already evaluate the next couple of maneuvers. This queue length is determined by the apps themselves.

Resulting API

Figure 9 summarizes our algorithm in the form of a resulting API. For an app to participate in VirtualSpace, it must implement this API.

```
abstract functions:
List<Vector2> ProvideProbabilityDistribution ()
List<TickInfo> ProvideTickInfo ()
void AdaptToMasterTicks(List<float> ticks)
List<Valuation> ValuatePotentialManeuver (
        List<Tile> maneuverEndTiles)
void ExecuteManeuver(Maneuver maneuver)

classes:
Tile{List<Vector2> area}
Frame{Tile tile, float time}
Maneuver{List<Frame> frames}
TickInfo{float tick, float variance}
Valuation{float weight,
        float preparation, float execution}
```

Figure 9: The VirtualSpace API

Contrasting to other approaches

Users behavior is not entirely predictable, as complex environments, such as games, offer various stimuli and users will make short-term decisions which cannot be perfectly predicted. Therefore, we did not follow high precision analytical planning approaches that assumes virtual humans (such as ORCA [4]), but instead used the described tile-based approach, which allows for variance and avoids deadlocks, where multiple users are crunched in one corner. Here the comparison to memory management comes in. Looking at different scheduling algorithms, one can see the resemblance to fixed-priority pre-emptive scheduling. Other algorithms, such as round-robin, would have resulted in lower utility for the apps (they cannot lead their users where they need them to be), or, such as shortest-remaining-time, cannot be used, as unpredictable variations in apps task times need to be taken into account.

DEMO APPLICATIONS

We have implemented four apps to run with VirtualSpace. Here we show how the API is implemented.

Badminton

A user plays badminton against an AI. The user sees the birdie's trajectory as soon as the AI hits, and an estimated trajectory before that. We display the trajectory at all times to avoid the natural tendency of users to walk towards the court's center. We made the virtual field slightly larger than the tracking area as users would optimize their movement and not walk onto the side of their field. When valuating maneuvers, badminton valuates larger areas stronger for quick ball exchanges that do not need to sync, also areas where the user has not yet hit that often, and areas with higher overlap to the court's service line. The minimum preparation time is the maximum time of a ball-exchange (dynamically computed), the minimum duration of the maneuver is based on the velocity of the birdie.

Pac-Man

As in the traditional Pac-Man, the user's goal is to collect all yellow pellets in a labyrinth. Ghosts position themselves so that the user does not pass into another user's area. Cherries provide additional points and thus draw users towards the next tile. They focus the user's attention using a distinct sound and a halo effect. When valuating maneuvers, Pac-Man valuates higher areas with more yellow pellets. The blue pill from Pac-Man serves as an in-app purchase (Figure 5); when collecting it, for a short duration the app valuations are generally higher than the ones of other applications. The sum of preparation time and maneuver duration is computed by the average player speed and is given to VirtualSpace as a dependency to be computed in a linear solver [9].

Space Invaders

As in the original arcade game, the user controls a spaceship to shoot enemy ships. Enemy ships are placed on all four sides of the environment. An area with protective blocks incentivizes the player to be in a certain area. Space Invaders valuates maneuvers that require the user to walk more. Preparation time uses a fixed delay and the duration is capped by the player's maximum walking speed.

Whac-A-Mole

The user is placed within a fenced area and is given a hammer to hit moles that spawn from the ground. Negative incentives, like flowers, are used to counteract the urge of users to walk towards the center. Audio cues given by the moles taunt the user to look towards them, in case the user faces the wrong way. Whac-A-Mole valuates areas higher where less moles have been hit. Preparation time and duration are similar to Space Invaders.

Application requirements

Planning and reacting

The design of apps is driven by two constraints: planning and reacting. VirtualSpace benefits from apps planning ahead of time and being able to react quickly. For a set of apps to run within the system, it must be assured that the apps react to the maneuvers that result from their own planning. Thus, the reaction time of the slowest application must be smaller to the planning time of the app least capable of planning. As an example, in our badminton app the worst possible time to perform a maneuver is right after the AI hits, as the app needs another full exchange (roughly two seconds), to influence the user again, so the reaction time is two seconds. It provides its valuations two exchanges ahead, taking roughly four seconds. If the badminton app was not able to do that, the birdie would not arrive where the app would want it to be. In Pac-Man, ghosts would need to travel too fast, etc.

Incentive considerations

Applications should always be able to place incentives inside the users' field of view, additionally to using audio cues. Also, both negative and positive incentives should be used – we found positive incentives (e.g., the birdie or moles) to work better for maneuvers, while negative incentives (e.g., the Pac-Man ghost) work better for keeping users within their area if no maneuver is active. If applications do not provide negative incentives, fallback routines from VirtualSpace happen more often, which lowers immersion.

Cognitive processing delay

Every app affords a different cognitive load to the user, so users' reaction time heavily depends on cognitive processing of the given incentives. For example, space invaders' protective obstacles start to move 400ms before the maneuver starts, giving the user a time to process and react. The other apps, which we also intentionally based on existing games to lower reaction times, are quite similar. In Pac-Man, the cherry, as a strong positive incentive, pops up seconds before the maneuver starts, but only when it does do the ghosts give way for the user to collect it. Pac-Man needs to allow for additional cognitive processing, as it includes more numerous game elements. We found iterative testing crucial for deducing the correct lead for placing incentives in the virtual environment, as various factors contribute to human reaction time [12].

Development and hardware

The applications were developed in Unity3D. The backend uses C# with the libraries Clipper [4], Protobuf [16] and Gurobi [9]. To allow researchers to replicate our work, we provided the full source code of VirtualSpace and the apps in C#/Unity3D [29]. The space is tracked using the Vive Lighthouse system with eight trackers [30]. The tracking information is forwarded via UDP to head-mounted displays (four GearVRs with Samsung S6 running Android) as shown in Figure 10.

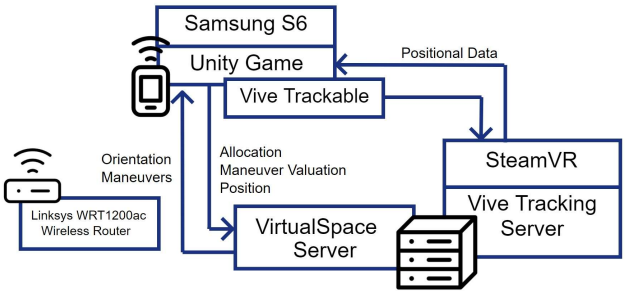

Figure 10: VirtualSpace's setup

USER STUDY

To better understand the resulting experience of allowing VirtualSpace to manage actual VR apps, we conducted a user study in which we compared VirtualSpace against the most commonly used approach, i.e., static pre-allocation of space ([1] uses a similar baseline). Our main hypothesis was that VirtualSpace provides more space coverage per user. Additionally, we assumed that VirtualSpace improves the experience, measured in ratings of confinement, enjoyment and presence.

Interface conditions

There were two space allocation conditions. In the *VirtualSpace* condition, we tested our system with the described maneuvers. In the *pre-allocated* condition each participant was confined to a static tile (no maneuvers).

We did not use *whole space allocation* as an additional baseline condition, in which participants take turns in using the whole space, as this assumes unlimited space for each app, which outperforms any technique.

In both conditions, participants walked in a 4x4m space, with a safety boundary of 60 cm between allocated areas. We deemed this boundary length sufficient after initial pilot studies.

Task and procedure

Participants were split into groups of four with each given one of the four games described above. Prior to the tasks, participants had 1 minute of training, in which they were playing their game alone in the tracking volume. Each group had two sessions, one for each space allocation condition. The order of conditions was counterbalanced. During the first session, participants played their app for five minutes, while the first space allocation strategy was applied. Participants then filled in a questionnaire about their experience containing two questions: "How much did you enjoy the experience?" and "How confined did you feel?" (Likert scale, 1-7) and the realism subscale of the presence questionnaire [32]. During the second session, they played the same app again for five minutes, while the other space allocation strategy was applied (within-subject design) and then again filled in another questionnaire. Each participant thus played their app twice, five minutes in each session, only using one app.

Participants

We recruited 16 participants from our organization (9 female, 7 male, age 28.8 ± 3.1 years), forming four groups. Seven participants had prior experience with VR (one had experienced real walking). The remaining nine participants had never tried VR before.

Results

As shown in Figure 11, users felt more confined in the pre-allocation condition, while VirtualSpace provided a greater sense of freedom ($p < .05$, $t(15) = -1.79$, one-sided). However, no statistically significant difference in enjoyment was observed ($p = .12$, $t(15) = 1.20$, one-sided).

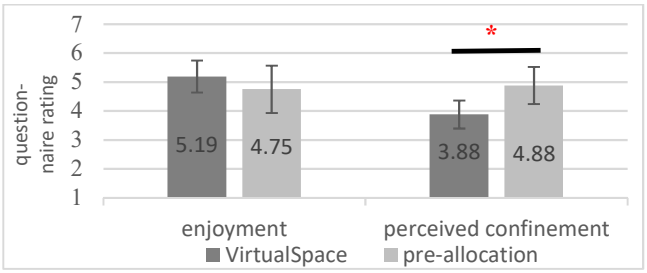

Figure 11: Participants felt more confined in the static pre-allocation condition, while VirtualSpace provides a greater sense of freedom.

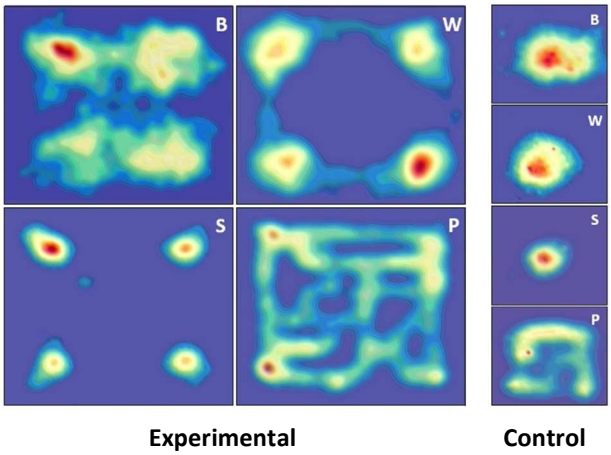

Figure 12: The four apps' space coverage for all participants (apps by initials). Participants covered more space using VirtualSpace than when using a static pre-allocation (axis lengths: 4m and 2m).

Space coverage was measured as all area being at least 30cm away from the tracked head mount throughout one session. Accounting all users, in the VirtualSpace condition 52 rotation maneuvers were conducted, 10 focus maneu-

vers, and 3 switch maneuvers. This led to significantly larger space coverage of 15.51 m^2 ± 2.01 per user in the VirtualSpace condition when compared to the space coverage of 3.46 m^2 ± 1.02 per user in the control condition (p < .001, t(15) = 21.12, one-sided). Figure 12 depicts these results.

For our analysis, we define a collision as a mutually unintended contact of two participants. We observed 7 collisions in total. Two collisions occurred in the first group, none in the second, two in the third and three in the fourth. We recorded differences between apps; the badminton app was involved in six collisions while the remaining apps were only involved in three collisions or less. All collisions occurred after a participant remained standing still when their tile moved; participants were absorbed in their experience, but did not follow the given incentive or the fallback visualization. Collisions occurred mostly during the first minutes of gameplay and further inspection revealed training effects; participants were less likely to breach (measured as the time ratio of being outside one's assigned tile, see Figure 13) if they had already experienced the app in the control condition (p < .05, t(14) = 2.28, one-sided).

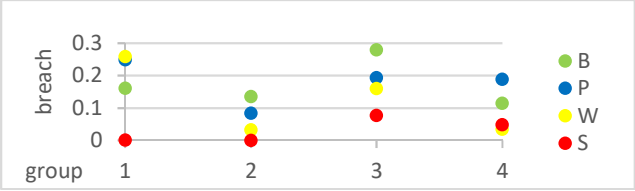

Figure 13: Participants compliance varied. Those who experienced the control condition first (group 2+4) were less likely to breach through their assigned areas, suggesting a training effect. Also, certain apps seem to keep their users within their tiles more effectively (apps by initials).

Despite participants occasionally experiencing the fallback mechanisms to keep them within their designated area, results did not show a significant difference in the realism subscale of the presence questionnaire [32] between conditions (p = .38, t(15) = 0.90, two-sided).

Qualitative feedback

Participants felt they had more space and made comments such as: "I just enjoyed having more space", "In the beginning [pre-allocation] it was not only more boring, but I felt way more cramped".

Participants remarked that they trusted the system, when apps kept participants within their assigned areas. However, when users did not recognize the incentive and reacted too late or apps needed to rely on the fallback to keep participants in place, participants' trust decreased: "The trust into the system was relatively high, that you do not walk into one another", "It then comes at quite a shock when they [fallbacks] pop up", "I actually felt safe – until that [collision] happened", "I felt I reacted too slowly".

Participants commented on the other users' participation. "Actually, I liked knowing that there were other people", "I would not play this alone, so it was actually even fun to maybe hit somebody".

DISCUSSION

Our main finding is that VirtualSpace outperforms static pre-allocation of space, as users feel less confined and can cover more space. This is true even for high user densities such as for our four users within 16m^2. This is an improvement over related work on space reduction techniques for real walking in VR, such as redirected walking [22] or dynamic layout generation [28], in which individual space requirements are of a far greater magnitude.

We believe our overloading technique can be used not only as an alternative, but also as a complement to these techniques. Researchers [1, 10] have already shown that redirected walking can apply to multiple users, while again using relatively large areas, which VirtualSpace can help to utilize more effectively. VirtualSpace on the other hand could use redirection techniques to orient users, instead of relying on in-game mechanics.

For our study we used some pre-existing metrics (collisions [1], presence [32]) and some that we specifically developed (perceived confinement, breach ratio, space coverage). To make their results comparable, we propose that future work also use these metrics.

The shown apps were VR typical isolating experiences. VirtualSpace can naturally be extended to multi-user applications, mobile scenarios, and spaces of different shape or size. Since primarily rotation maneuvers have been carried out, a more even coverage of space still seems possible. Increasing the size of the tracking space while maintaining user density could also improve ratings, as the effect of real walking might prove higher in larger areas on enjoyment and perceived confinement.

We acknowledge that VirtualSpace poses a certain physical risk, so further research into safety and trust is required. A total of seven collisions occurred. Related work [1] achieved a reduction of collisions of over 58% in a simulated framework with much lower user density – although we have not conducted a collision baseline due to safety risks, we assume this compares well, but agree that no collisions are more desirable. Based on our qualitative analysis and findings in literature, we can argue that risk contributed to positive experiences for some participants as "too close a distance" between users can be an intriguing game element [14]. However, enjoyment also relies on trust and participants' remarks suggest an individually perceived lack of trust, which depends on apps being able to prepare their user for a maneuver and keep them within their assigned areas to avoid breaches. This would explain our findings in the enjoyment question and realism subscale. We expected the experimental condition to perform measurably higher, which might be attributed to an individually perceived lack of trust. Having users with different mindsets or player

types [2] might pose an additional challenge here, as explorative and/or social types might deliberately walk into unintended areas and frequently interrupt others, which happened with some users during our study. The boundary size of 60cm, as determined during initial pilot studies, did not seem to be a main factor for breaches, the biggest challenge instead is to ensure that users notice the given incentive. Research on user visualizations [13, 17] in shared space already addresses this issue in part. As we discovered training effects, future investigation into safety and trust should encompass factors such as adaptive maneuver rates to reduce breaches and also maneuver interrupts when users fail to notice their incentive or do not comply to it.

VirtualSpace helps minimize individual space requirements. This will help real walking applications expand to different scenarios such as mobile VR, where space needs to be shared with other people, or application domains requiring space with high interaction rates, such as e-sports, as demonstrated with our demo apps.

CONCLUSION

We presented VirtualSpace, a technique for overloading multiple real walking VR users into the same physical space. VirtualSpace achieves this by containing each app in a smaller tile. Frequent "maneuvers" allow apps to incentivize their users to walk across the entire physical space, thereby allowing each app to progress its narrative and to prevent users from noticing that they are confined to a tile. This strategy enables VirtualSpace to achieve packings of the unprecedented density of four users in $16m^2$ as we demonstrated in our user study.

REFERENCES

1. Mahdi Azmandian, Timofey Grechkin, and Evan Suma Rosenberg. An evaluation of strategies for two-user redirected walking in shared physical spaces. In *Virtual Reality (VR)*, Los Angeles, CA, pp. 91-98. https://doi.org/10.1109/VR.2017.7892235
2. Richard Bartle. 1996. Hearts, Clubs, Diamonds, Spades: Players who suit MUDs. *Journal of MUD research.* 1(1), 19-58.
3. Blaine A. Bell and Steven K. Feiner. 2000. Dynamic space management for user interfaces. In *Proceedings of the 13th annual ACM symposium on User interface software and technology* (UIST '00), 239–248. https://doi.org/10.1145/354401.354790
4. Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n-body collision avoidance. In *Robotics Research - The 14th International Symposium ISRR*, Springer Tracts in Advanced Robotics, vol. 70, Springer-Verlag, May 2011, pp. 3-19. https://doi.org/10.1007/978-3-642-19457-3_1
5. Clipper, Open source freeware library. Retrieved September 17, 2017 from http://angusj.com/delphi/clipper.php
6. Evren Bozgeyikli, Andrew Raij, Srinivas Katkoori, and Rajiv Dubey. 2016. Point & Teleport Locomotion Technique for Virtual Reality. In *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play* (CHI PLAY '16), 205-216. https://doi.org/10.1145/2967934.2968105
7. Rudolph P. Darken, William R. Cockayne, and David Carmein. 1997. The omni-directional treadmill: a locomotion device for virtual worlds. In *Proceedings of the 10th annual ACM symposium on User interface software and technology* (UIST '97), 213-221. http://doi.org/10.1145/263407.263550
8. Brett R. Fajen and William H. Warren. 2003. Behavioral dynamics of steering, obstacle avoidance, and route selection. *Journal of Experimental Psychology. Human Perception and Performance* 29(2), 343–362. https://doi.org/10.1167/1.3.184
9. Gurobi, Linear Solver, Academic License. Retrieved September 17, 2017 from http://www.gurobi.com
10. Jeannette E. Holm. 2012. Collision prediction and prevention in a simultaneous multi-user immersive virtual environment. PhD diss., Miami University. Retrieved January 6, 2018 from https://etd.ohiolink.edu/
11. Victoria Interrante, Brian Ries and Lee Anderson. 2007. Seven League Boots: A New Metaphor for Augmented Locomotion through Moderately Large Scale Immersive Virtual Environments. In *Symposium on 3D User Interfaces*, Charlotte, NC, 2007, pp. https://doi.org/10.1109/3DUI.2007.340791
12. Robert J. Kosinksi. 2013. *A Literature Review on Reaction Time.* Clemson University.
13. Jérémy Lacoche, Nico Pallamin, Thomas Boggini, and Jérôme Royan. 2017. Collaborators awareness for user cohabitation in co-located collaborative virtual environments. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology* (VRST '17). ACM, New York, NY, USA, Article 15, 9 pages. https://doi.org/10.1145/3139131.3139142
14. Florian Mueller, Sophie Stellmach, Saul Greenberg, Andreas Dippon, Susanne Boll, Jayden Garner, Rohit Khot, Amani Naseem, and David Altimira. 2014. Proxemics play: understanding proxemics for designing digital play experiences. In *Proceedings of the 2014 conference on Designing interactive systems* (DIS '14), 533-542. https://doi.org/10.1145/2598510.2598532
15. Tabitha C. Peck, Henry Fuchs, and Mary C. Whitton. 2010. Improved Redirection with Distractors: A large-scale-real-walking locomotion interface and its effect on navigation in virtual environments. In *Proceedings of IEEE Virtual Reality Conference* (VR'10), 35-38. https://doi.org/10.1109/VR.2010.5444816

16. Protocol Buffer, Contract Based Serializer. Retrieved September 17, 2017 from https://github.com/mgravell/protobuf-net
17. Anthony Scavarelli and Robert J. Teather. 2017. VR Collide! Comparing Collision-Avoidance Methods Between Co-located Virtual Reality Users. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (CHI EA '17), 2915-2921. https://doi.org/10.1145/3027063.3053180
18. Keng Hua Sing and Wei Xie. 2016. Garden: A Mixed Reality Experience Combining Virtual Reality and 3D Reconstruction. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (CHI EA '16), 180-183. https://doi.org/10.1145/2851581.2890370
19. Misha Sra and Chris Schmandt. 2015. MetaSpace: Full-body Tracking for Immersive Multiperson Virtual Reality. In *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (UIST '15 Adjunct), 47-48. https://doi.org/10.1145/2815585.2817802
20. Misha Sra. 2016. Asymmetric Design Approach and Collision Avoidance Techniques For Room-scale Multiplayer Virtual Reality. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16 Adjunct), 29–32. https://doi.org/10.1145/2984751.2984788
21. Misha Sra, Sergio Garrido-Jurado, Chris Schmandt, and Pattie Maes. 2016. Procedurally generated virtual reality from 3D reconstructed physical space. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology* (VRST '16), 191-200. https://doi.org/10.1145/2993369.2993372
22. Sharif Razzaque, Zachariah Kohn, and Mary C. Whitton. 2001. Redirected walking. In *Proceedings of EUROGRAPHICS* 9, 105-106.
23. Steam VR Survey. Retrieved September 17, 2017 from https://steamcommunity.com/app/358720/discussions/0/350532536103514259/?ctp=2#c1332580922532225 57
24. James N. Templeman, Patricia S. Denbrook and Linda E. Sibert. 1999. Virtual Locomotion: Walking in Place through Virtual Environments. In *Presence* 8(6), 598-617. https://doi.org/10.1162/105474699566512
25. Martin Usoh, Kevin Arthur, Mary C. Whitton, Rui Bastos, Anthony Steed, Mel Slater, and Frederick P. Brooks, Jr. 1999. Walking > walking-in-place > flying, in virtual environments. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '99), 359-364. https://doi.org/10.1145/311535.311589
26. Measuring Actual Use of Space. Retrieved March 3, 2017 from http://agilquest.com/Whitepapers-Reference-Guides-and-Checklists/measuring-actual-use-of-space.html
27. OfficeSpace. Software for space management in an office. Retrieved March 3, 2017 from https://www.officespacesoftware.com/
28. Khrystyna Vasylevska, Hannes Kaufmann, Mark Bolas and Evan A. Suma. 2013. Flexible spaces: Dynamic layout generation for infinite walking in virtual environments. In *2013 IEEE Symposium on 3D User Interfaces* (3DUI'13), 39–42. https://doi.org/10.1109/3DUI.2013.6550194
29. VirtualSpace, Online Repository. Retrieved January 8, 2018 from https://github.com/HPI-VirtualSpace
30. Vive and Vive Trackers. VR controller. Retrieved September 17, 2017 from https://www.vive.com/us/vive-tracker/
31. Betsy Williams, Gayathri Narasimham, Bjoern Rump, Timothy P. McNamara, Thomas H. Carr, John Rieser, and Bobby Bodenheimer. 2007. Exploring large virtual environments with an HMD when physical space is limited. In *Proceedings of the 4th symposium on Applied perception in graphics and visualization* (APGV '07), 41-48. https://doi.org/10.1145/1272582.1272590
32. Bob G. Witmer and Michael J. Singer. 1998. Measuring Presence in Virtual Environments: A Presence Questionnaire. *Presence: Teleoperators and Virtual Environments* 7(3), 225-240. http://doi.org/10.1162/105474698565686
33. P. P. Zouein and I. D. Tommelein. 1999. Dynamic Layout Planning Using a Hybrid Incremental Solution Method. *Journal of Construction Engineering and Management.* 125(6), 400-408. https://doi.org/10.1061/(ASCE)0733-9364(1999)125:6(400)