Creating and Augmenting Keyboards for Extended Reality with the Keyboard Augmentation Toolkit

MARK MCGILL, University of Glasgow, Scotland, UK
STEPHEN BREWSTER, University of Glasgow, Scotland, UK
DANIEL PIRES DE SA MEDEIROS, University of Glasgow, Scotland, UK
SIDNEY BOVET, Logitech S.A., Switzerland
MARIO GUTIERREZ, Logitech S.A., Switzerland
AIDAN KEHOE, Logitech Design Lab, Ireland

This paper discusses the *Keyboard Augmentation Toolkit* (KAT) which supports the creation of virtual keyboards that can be used both for standalone input (e.g. for mid-air text entry) and to augment physically tracked keyboards/surfaces in mixed reality. In a user study, we firstly examine the impact and pitfalls of visualising shortcuts on a tracked physical keyboard, exploring the utility of virtual per-keycap displays. Supported by this and other recent developments in XR keyboard research, we then describe the design, development and evaluation-by-demonstration of KAT. KAT simplifies the creation of virtual keyboards (optionally bound to a tracked physical keyboard) that support *enhanced display* - 2D/3D per-key content that conforms to the virtual key bounds; *enhanced interactivity* - supporting extensible per-key states such as tap, dwell, touch, swipe; *flexible keyboard mappings* that can encapsulate groups of interaction and display elements e.g. enabling application-dependent interactions; and *flexible layouts* - allowing the virtual keyboard to merge with and augment a physical keyboard, or switch to an alternate layout (e.g. mid-air) based on need. Through these features, KAT will assist researchers in the prototyping, creation and replication of XR keyboard experiences, fundamentally altering the keyboard's form and function.

CCS Concepts: • **Human-centered computing → Human computer interaction (HCI)**; **Virtual reality**; **User interface toolkits**; **Empirical studies in HCI**; **Keyboards**.

Additional Key Words and Phrases: Mixed Reality; Virtual Reality; Augmented Reality; Augmented Keyboards; Keyboard Input; Shortcuts;

ACM Reference Format:
Mark McGill, Stephen Brewster, Daniel Pires De Sa Medeiros, Sidney Bovet, Mario Gutierrez, and Aidan Kehoe. 2022. Creating and Augmenting Keyboards for Extended Reality with the Keyboard Augmentation Toolkit. *ACM Trans. Comput.-Hum. Interact.* 29, 2, Article 15 (April 2022), 38 pages. https://doi.org/10.1145/3490495

1 INTRODUCTION

The physical keyboard has been the *de facto* peripheral for interaction for decades due to the high performance in both text entry and shortcut usage that can be achieved. Accordingly, the physical keyboard layout and controls have been slow to change in the face of the emergent challenges users now face. Modern applications provide many keyboard shortcuts, more than 100 in Microsoft Office, yet despite efforts to accommodate learning and discovery [34, 61], these shortcuts remain largely unused. One reason is that they can be hard to remember and the keyboard cannot display a congruent visual indication to help. Moreover, keyboards are still predominantly

Authors' addresses: Mark McGill, University of Glasgow, Scotland, UK, mark.mcgill@glasgow.ac.uk; Stephen Brewster, University of Glasgow, Scotland, UK, stephen.brewster@glasgow.ac.uk; Daniel Pires De Sa Medeiros, University of Glasgow, Scotland, UK, Daniel.PiresdeSaMedeiros@glasgow.ac.uk; Sidney Bovet, Logitech S.A., Cork, Switzerland, sbovet@logitech.com; Mario Gutierrez, Logitech S.A., Lausanne, Switzerland, mgutierrez1@logitech.com; Aidan Kehoe, Logitech Design Lab, Cork, Ireland, akehoe@logitech.com.

limited to binary keypress input, lagging behind On Screen Keyboards (OSKs) on touch surfaces. For example, the touchscreen user can switch from alphanumeric input to direct GUI interaction without necessitating a costly change in input modality [66], in a way that the physical keyboard user cannot. The physical keyboard also lags behind in capability for expression for the same reason - despite efforts from the likes of Microsoft [107], adding an emoji [82] into a message can still prove problematic compared to using OSKs. For productivity outside the home or office (e.g. in a coffee shop or train), the form factor and expectations regarding keyboard functionality are such that we have not yet been able to move toward a "minimal" portable keyboard design, minimising the number of keys by appropriating keys on demand to support transient functions such as volume control.

However, the constraints regarding physical keyboard form and function can be overcome through the use of Mixed Reality (MR) headsets. These headsets are improving in terms of fidelity, cost, sensing and form factor. Their adoption will inevitably drive a transition from physical to virtual displays [23] and more spatial computing. MR headsets can augment tracked objects [9], peripherals and surfaces [8, 42], meaning a physical keyboards form and function could be altered/augmented by such headsets [12]. This is of increasing relevance given the potential for MR to enhance productivity environments in particular, demonstrated both in research [28, 65, 78] and by industry, for example through the Oculus "infinite office" concept [76].

Augmented Keyboards are physical keyboards that can be positionally tracked by a MR headset, such that aligned virtual augmentations can be rendered on or around them. This concept allows for re-mapping of both input and output. Research has explored many concepts in this space, such as supporting secure typing in AR [60] or using the keyboard as a haptic proxy [90] for a variety of interfaces. However, such implementations are typically highly bespoke and closed source. This hinders extension of, and comparison to, prior research.

This paper describes the steps we have taken toward resolving this issue through the development of the novel, open source Keyboard Augmentation Toolkit (KAT) for Unity, which facilitates prototyping of virtual and augmented keyboards for XR/MR. Using an early version of KAT, we first examine the usability of a basic augmented keyboard experience, looking at the impact and pitfalls of visualizing shortcuts. We also reflect on advances in consumer-oriented augmented keyboards, discussing the progress made with the *Logitech MR*

Fig. 1. A live screenshot of the Logitech MR keyboard using our Keyboard Augmentation Toolkit, rendered in VR with Mixed Reality video-passthrough hand segmentation and rendering.

Keyboard, building on previous developments such as the Logitech Bridge SDK [11, 103] in supporting a VR-trackable keyboard, and the Oculus infinite office integration of optical tracking of the Logitech k830 keyboard [77] - both significant steps toward a consumer-oriented MR keyboard. We then discuss the current iteration of KAT, whose capabilities have been influenced both by the preceding research in this paper, as well as seminal research from the likes of Block *et al.* [10], Gellersen *et al.* [32] and Schneider *et al.* [90].

KAT supports two key features: the creation of *virtual keyboard augmentations*, and the separate definition of *virtual keyboard layouts*. For augmentations, KAT effectively allows for every key to be treated as an interactive display, supporting generic hand-tracked interactions (e.g. tap, swipe, hover), flexible display options per-key (e.g. 2D/3D icons, unicode text), importing existing Windows/Linux keyboard mappings, hierarchical mappings (enabling multiple applications to share usage of the augmented keyboard overlay) and an event infrastructure which allows for easy extensibility to support novel interactions in the future. For layouts, KAT supports both imported 2D layouts from Keyboard Layout Editor [85], and modelled/3D layouts, with layouts able to be changed on-the-fly, for example allowing a virtual keyboard to dock with a tracked physical keyboard when the user sits at their desk, conforming to the physical keyboard layout; and then subsequently change to an appropriate mid-air interaction layout when the user moves away from the desk once more. Importantly, KAT supports the binding of the virtual keyboard augmentations to the current layout dynamically, meaning that any augmentations designed for KAT will work on the majority of envisioned 2/3D layouts. In this way, we provide an architecture for prototyping varied interactions for both virtual augmentations of physical keyboards, and other mid-air or surface-aligned virtual keyboards, supporting exploration of the relationship between physical and mid-air keyboards in MR. In an evaluation-by-demonstration [53] of our toolkit, we outline how practitioners can create novel layouts, mappings and interactions by reproducing and extending prior research.

Our intention with KAT is to empower the HCI community to better guide the requisite standards, interaction designs, and architecture of future XR / MR keyboards, and facilitate comparisons between novel and past keyboard layouts and interactions through encouraging their development and replication on a common platform. In particular, where previous research has been closed source and bespoke, KAT democratises our ability to explore new augmented keyboard interactions and virtual keyboard layouts, in a way that will better support rapid prototyping, and open-source collaboration and sharing of implementations.

1.1 Contribution Statement

This paper contributes to our understanding of mixed reality keyboard interactions in three key ways:

(1) A user study evaluating the most immediate and widely recognised potential use case of augmented keyboards, *shortcut visualization*. We provide novel insights into the efficacy, benefits and pitfalls of this capability. We also reflect on the implications such a capability has for how we can support augmentations specific to applications and their current context.
(2) Informed by this study, we contribute the *Keyboard Augmentation Toolkit* (KAT) for prototyping keyboards for MR. KAT supports a breadth of augmentations (both 2D and 3D); extensible support for a variety of on/around key interactions; and an architecture that separates the keyboard layout from the applied augmentations, consequently enabling the creation of augmentations and mappings that can be utilized with both mid-air and surface/physical keyboard aligned layouts. This toolkit will enable practitioners to more rapidly prototype novel MR keyboard layouts and augmented keyboard interactions, and more easily share their implementations, aiding direct comparisons in subsequent research.
(3) To evaluate our toolkit, we follow an evaluation-by-demonstration approach (similarly to [53]) by exemplifying its capacity to support both new interactions, as well as simplifying replication of prior research such as existing keyboard mappings (e.g. the new AZERTY [25–27]); layouts for both 3D mid-air [111] and 2D

surface-aligned [22] virtual keyboards; and per-key interactions (e.g. GestAKey swipes [93], Metamorphe [6] tilting keys).

2 RELATED WORK

2.1 Physical Augmentations of Keyboards

2.1.1 Visual Augmentations. Visual feedback incorporated into physical keyboards has been repeatedly explored. Consumer products have incorporated additional displays co-located with buttons (e.g. Logitech G19 [58]), sometimes with touch input (e.g. Razer DeathStalker [86] or Apple Touch Bar [2]). Per-key RGB lighting has become increasingly common in gaming keyboards, being used both to delineate shortcuts and provide visual feedback on in-game events. Keycap displays have been repeatedly explored, from the OLED-driven Optimus concept from Art. Lebedev of a decade ago [98] to more recent e-ink efforts such as Nemeio [72], but have not been commercially successful. In research, DisplayCover [35] was a precursor of the Apple Touchbar, where a visual touch surface was used to replace the function key bar on laptops). In this case, Windows Live tiles were displayed on a touch surface above the keyboard to allow for quick touch and gesture interactions when using a laptop. Ephemeral interactions [106] appropriated existing physical surfaces for touch input, for example overlaying a physical slider interface on the unused surface of a keyboard. Magic desk [8] augmented the surfaces around the keyboard and mouse, offloading functionality and visualizing shortcuts on surfaces that were quick to transition to from keyboard/mouse usage, a theme that has frequently occurred [42, 104, 110, 116]. ControllAR [7] augmented control surfaces with visual feedback overlaid on top of both the surface and the fingers/hands using a transparent display positioned between the surface and the user, with the visual feedback comprising of user-selected portions of existing applications for GUI "remixing". Notably, whilst the space around the keyboard has been repeatedly visually augmented, the space above the keyboard (i.e. within reach of fingertips without moving hands off the keyboard and onto the desk surface) has not. Augmented keyboards can utilize this space thanks to head-tracked rendering.

2.1.2 Training and the Case for Visual Feedback of Keyboard Shortcuts. The benefits of keyboard shortcuts have been a source of significant debate, with various studies showing both performance benefits over GUIs [79] and detriments [101], differences in terms of the perception of efficiency [52, 99], and even a lack of difference regarding performance/perceived usability [91]. However, keyboard shortcuts appear beneficial to power users, particularly where costly transitions between keyboard and mouse are required [66], with caveats when keyboard shortcut actions are hierarchical (i.e. requiring shortcut sequences) [66]. For "heavily-used interfaces, keyboard shortcuts can be as efficient as toolbars and have the advantage of providing fast access to all commands" [79].

The problem has been in ensuring uptake of shortcuts over less efficient but more immediately accessible methods such as menus. Ni *et al.* [73] noted that shortcuts are difficult to learn, lack consistency (e.g. different actions might have different shortcuts across applications) and lack visibility. Demonstrating this, a previous study (n=38) [39] looking at shortcut knowledge in Microsoft Word found that of the 16 most common actions, experts correctly identified 81% of the associated keyboard shortcuts, with novices approaching 63%. Lane *et al.* noted that keyboard shortcuts are more efficient than equivalent GUI shortcuts, however experienced users rarely use efficient keyboard shortcuts, instead favouring GUI equivalents [52]. They concluded that users needed support in transitioning from menus and toolbars to keyboard shortcuts, for example through the use of training: "people do not always use the optimal method, the method they use is probably 'good enough' for them... [bad] habits caused by this satisficing behavior quickly form and are hard to change". They also demonstrated that prompting regarding keyboard shortcuts greatly increased their uptake.

Commonly, training is employed to build the user's knowledge of salient controls. The Hotkey Palette [1] visualized shortcuts and document actions for an OSK. ExposeHK [62] (feedforward, presented prior to action) showed keyboard shortcuts in GUI menus and ribbons when the modifier key was pressed, a feature that has

seen some adoption in applications, whilst [91] employed early notifications to increase awareness of shortcuts. Projects such as Hotkey-Eve [96] and HotkeySkillometer [63] used feedback to convey shortcuts that could have been used. IconHK (Giannisakis *et al.*) [34] blended visual cues regarding keyboard shortcut information into toolbar GUI elements that adapted based on prior usage and system intention to provoke reminders of shortcuts.

However, fundamentally, there are advantages to facilitating visual search, with Scarr *et al.* [89] noting that "novices benefit from using visual search for salient controls, rather than retrieving command names from memory or manuals" (based on [94]). Scarr *et al.* defined a framework of necessary features for expertise development in shortcut usage [18, 89] - of note here were the "visibility and ready-to-hand" and "low display demands" factors, both of which are satisfied by an augmented keyboard. Giannisakis *et al.* [34] identified similar key design challenges in conveying keyboard shortcuts: maximizing shortcut exposure duration, minimizing the visual space used to convey the shortcuts, conveying the meaning of the commands through iconography, and maintaining the aesthetic appeal of the icons. It has been shown that keyboard shortcut usage in-part exhibits elements of satisficing behaviour [99]. Pertinently, it was suggested that to break this behaviour "the optimal methods might better be learned as soon as possible, by, for example, making (shortcuts) more salient".

2.1.3 Supporting Interactions On and Around the Keyboard. Various projects have expanded the capabilities of keyboard input. Finger-aware shortcuts exploited finger tracking to allow a single keypress to have multiple command mappings, whilst FingerArc and FingerChord provided visual guidance on-screen regarding the available commands [114, 115]. GestAKey allowed for multiple actions being associated to a keystroke through touch sensitive keycaps [93], an approach that has repeatedly been explored [92, 116]. Other gesture approaches for extending keyboard shortcuts [14] such as gesturing on/above the surface of the keyboard have also been examined [100, 113]. More recently, touch sensitivity has been explored for keyboards in AR/MR [80], suggesting new possibilities for per-key states driven by hand tracking technology in particular.

Whilst these interaction techniques may have been validated for performance, the discoverability of the functionality could be problematic; users are entirely reliant on off-device visual feedback [30] to guide them regarding the interaction. Moreover, these techniques have typically been implemented using bespoke sensing solutions. However, the advent of augmented keyboards could resolve these issues, allowing for on/around key rendering of functionality and potentially improving the discoverability of these interactions. For example, resting a finger on a GestAKey or Metamorphe [6] key could reveal mid-air tooltips indicating the available actions. The possibility of augmented keyboards suggests prior interactions that were not feasible for mass deployment could be re-appraised and deployed on a real-world scale, driven by XR headset sensing.

2.1.4 Touch-Display Keyboards. An augmented keyboard of particular note, *Touch-Display Keyboards* (TDKs) [10, 32] used capacitive touch sensing and overhead projection to prototype novel interactions and visual augmentations of a keyboard. Block *et al.* discussed the design space of their keyboard display in terms of a number of potential interactions/features made possible. For example, they offered up designs for rendering content on keys to visualize hidden functionality, switching between keyboard mappings (e.g. based on different languages), and toggling keyboard maps based on dedicated hotkeys. They also discussed multi-key buttons, temporary menus/widgets and scalar controls of variables (e.g. volume). Possible key states were expanded upon, with touch events being exposed as events that could for example preview shortcut tooltips or provide new overloaded actions. However, no formal user evaluations were reported, aside from limited user comments. Thus, the potential impact of key features such as shortcut visualization currently lacks evidence/practical insights regarding usage. The TDK system was also closed source and thus not replicable/extendable, but formed the foundation of our thinking for KAT.

Incorporating physical displays into keyboards can also be problematic: cost and power requirements alone were enough to ensure that the Optimus concept was not successful, with recent efforts using e-ink technology [72] still far from the capabilities imagined. Keyboards with integrated displays are inherently limited: fidelity is

bounded by the specifications of the integrated keyboard display (rather than the changing HMD if rendered in MR), nor can they render with depth around/above the keyboard. Such integration is also problematic from an environmental perspective, as the keyboard becomes yet another component to be periodically upgraded and discarded.

2.2 Using Physical Keyboards In MR

A variety of interaction techniques have been devised for text entry specifically in VR (e.g. [97, 109]). However, the physical keyboard, when rendered in some fashion in VR along with the user's fingertips, remains the best performing technique. McGill *et al.* [64] first discussed bringing the physical keyboard into VR, using a video-pass through technique to selectively render both the keyboard and hands based on user engagement. Subsequently, this work has been extended both in research [36–38, 44, 48, 56, 105] and commercially, for example with the Logitech Bridge SDK [11, 103]. These typically incorporate both positional tracking of the physical keyboard to allow for congruent rendering of a virtual model in VR, and some form of finger/hand tracking to allow for different representations of fingertips and hands in VR. With respect to positionally tracked keyboards, whilst Bovet *et al.* showed that a video pass-through approach for hand rendering gave the best performance, [11, 37, 38, 48] found that fingertip tracking was sufficient for good text entry in VR, whilst [48] found similar efficacy for fingertip visualization in experienced typists. Hoppe *et al.* [44] showed that with a Leap Motion tracking performance could reach 71% of baseline.

Keyboard augmentations and the use of physical keyboards in VR [12] has also previously been explored, building upon the work of Block *et al.'s* TDKs. Simeone *et al.* [95] discussed rendering different skins on the keyboard to preserve presence in immersive virtual environments. Maita *et al.* [60] examined secure text entry in AR by augmenting physical keyboards with a randomized layout. McGill *et al.* utilized a Logitech MR keyboard in a virtual workspace environment, augmenting individual keys to denote functions for controlling the position of surrounding virtual displays [65]. And *ReconViguRation* [90] broadly explored a range of potential use cases of an augmented keyboard, specifically how the physical keyboard could be reconfigured in VR to support emoji entry, special characters, secure password entry, foreign languages, browser shortcuts, text macros and passive haptics. They captured ratings of ease of use, utility and enjoyment of the discussed concepts, finding ease of use in particular to be high across the envisioned use cases. However, as with the TDK paper, the implementation was closed source, and focused on the design space, rather than the mechanics of supporting/defining/managing per-key augmentations. They concluded that *"our work raises questions about how to best design for perceived affordance [and] how to best dynamically reconfigure a keyboard"*. We suggest this in particular is a key motivating point - whilst there is likely to be a near-limitless variety of use cases where a keyboard could beneficially be augmented, we currently lack the practical knowledge regarding how to logistically facilitate this. Satisfying such questions is particularly difficult without re-usable tools to better support practitioners in the creation and exploration of dynamic, reconfigurable keyboards. This is a point that we address through KAT. Finally, ControllAR examined rendering augmentations on MIDI controller buttons, finding that "they are often preferred to touchscreens because of the variety of gestures that the sensors allow and because of the haptic feedback that they provide, and to tabletops with tangibles because of their compactness" [7].

2.3 The Possibilities of Augmented Keyboards

By augmenting an existing positionally tracked peripheral such as a physical keyboard using an MR headset, we can avoid tightly coupling the peripheral to integrated display(s), such as in the case of Optimus or Nemeio. If we assume MR headsets will eventually form the basis of mobile spatial computing (e.g. AR headsets rendering virtual displays in mid-air, VR immersive gaming, etc.) then we can begin to design interactions that take advantage of the sensing capabilities of these headsets. We might utilize the ability to render with depth on/around/above the

keyboard combined with hand/finger tracking to support keyboard-initiated mid-air interactions e.g. creating blended reality UIs for VR/AR 3D modelling. More generally, such a vision would also facilitate better transfer of HCI-led advancements into consumer contexts. For example, GestAKey [93] could in-part be implemented virtually, by tracking finger position around/on specific keys, without requiring bespoke hardware. FingerArc and FingerChord [114, 115] could also be enacted entirely using MR headset sensing, with visual guidance now rendered *in situ* rather than on a separate display off-device [30]. Other gesture approaches for extending keyboard shortcuts [14] and gesturing on/above the surface of the keyboard [100, 113] could equally be supported with congruent visuals, effectively democratising access to a variety of previously envisaged novel keyboard interactions.

3 STUDY: ASSESSING SHORTCUT DISCOVERABILITY

Motivated by our literature review, we built an initial prototype of KAT, targeting the generic definition and rendering of 2D augmented keyboard overlays. KAT initially evolved from efforts within Logitech regarding their Bridge SDK [103] for the HTC Vive, a proof of concept (influenced by our prior work [64]) demonstrating how a physical keyboard could be tracked and integrated into VR using a Vive Tracker. Importantly, Logitech also utilized the front-mounted camera of the HTC Vive alongside this positional tracking to render video pass-through hand tracking. This development indicated that a tracked, high-throughput mixed reality augmented keyboard was now technologically feasible, prompting our research.

The most commonly envisioned use case for keyboards with per-key displays (both physical and virtual) is that of per-keycap shortcut visualization. However, little is known regarding the impact, efficacy and pitfalls of visualizing shortcuts for an existing, known application, outside of anecdotal findings. TDKs [10] first suggested the visualization of shortcuts on the keyboard surface, and others have followed suit [90], but the efficacy of the approach has never been evaluated. Our intention was to use this prototype to assess per-key shortcut discoverability, addressing the following research question: **RQ1:** *Does visualizing unknown shortcuts on an augmented keyboard make them more discoverable?*

Fig. 2. Left: Experimental setup, with 3 Optitrack 13W cameras tracking markers attached to the participant's fingers, as well as rigidbodies attached to the keyboard/VR headset. Right: View in VR, with a virtual display showing the Windows desktop and an aligned virtual representation of the physical keyboard.

3.1 Tracked Keyboard Implementation

Our implementation utilized VR Desktop Mirror [17] for rendering the Windows desktop in Unity3D, alongside three Optitrack 13W cameras for keyboard and fingertip tracking. An i7 PC with a nVidia GTX970 was used for rendering all conditions, with a Samsung Odyssey Microsoft Mixed Reality VR headset [88] used throughout (1440x1600 pixels per eye). To align the Optitrack and headset coordinate systems, a rigid body attached to the headset was tracked for one frame on application startup and used to transform the Optitrack coordinate space. Optical markers (6.4mm, M3) were attached to the participant's fingernails to visualize fingertip locations (see Figure 2), with fixed offsets used to correct marker positions to appear as fingertip positions in VR. For the physical keyboard, we used a Logitech G810 with the 3D model taken from the Logitech Bridge SDK [103], with optical markers attached in a known rigid body configuration for positional tracking. It should be noted that we did not rely on the Logitech Bridge SDK directly for this implementation as calibration issues/tracker drift were such that the digital twin of the keyboard could become mis-aligned with reality (an issue subsequently solved by the Logitech MR keyboard discussed later in the paper). See the video figure for footage in action.

3.2 Design and Demographics

We conducted our evaluation in two parts - an initial assessment of baseline and VR typing performance to provide context regarding our tracked keyboard implementation (for example, to what extent errors in keyboard usage might contribute to apparent errors in shortcut selection), followed by the shortcut discoverability evaluation. For all results, a repeated measures ANOVA was performed. Where data were non-parametric, an Aligned-Rank Transform [108] was used to allow parametric methods. For effect size, Generalized Eta Squared (η_g^2) is reported (see [20] for interpretation), as well as the BayesFactor (BF^{10}, calculated using the *BayesFactor* R package, see [46] for interpretation). For *post hoc* contrasts, the *lsmeans* [55] R package was used with Tukey adjustment. The box plots feature notches denoting the 95% confidence level [49]. 18 participants were recruited from University mailing lists (mean age=28.1±5, 9 male, 9 female) to take part, and were paid £10 for their time. Participants experienced all tasks (with counter-balanced condition ordering throughout).

3.3 Part 1: Baseline Performance

To contextualise the typing performance of our Optitrack-based VR keyboard and fingertip tracking, we used a text entry task - 8 phrases per condition after training (typing until the user felt as comfortable as they felt possible) using the Mackenzie 500 phrase set [59], enacted using WebTEM [4]. The aim was to get a baseline of typing performance with an augmented keyboard, and in particular the accuracy of selection for individual key presses, to validate that our prototype implementation was sufficient for examining visualized shortcut usage.

Fig. 3. Baseline performance keyboard layouts in VR. Left: standard QWERTY. Right: randomized.

There were three conditions: a **Reality** baseline where users typed on the keyboard in reality using a monitor to display the text; **VR** where users performed the same task in our VR workspace; and **Random** where users performed the same task in VR but using a randomized keyboard layout (see Figure 3), intended to test their

accuracy in selecting the desired key whilst discounting touch typist muscle memory. Performance metrics (WPM, Total Error Rate) were recorded for the text entry task, and a smartphone-based questionnaire after each condition recorded select NASA TLX [41] subscales (Performance, Effort and Frustration).

There were significant effects (see Table 1 and Figure 4) on task metrics (WPM, total error rate - the character error rate) and subjective measures (TLX Performance, Effort, Frustration).

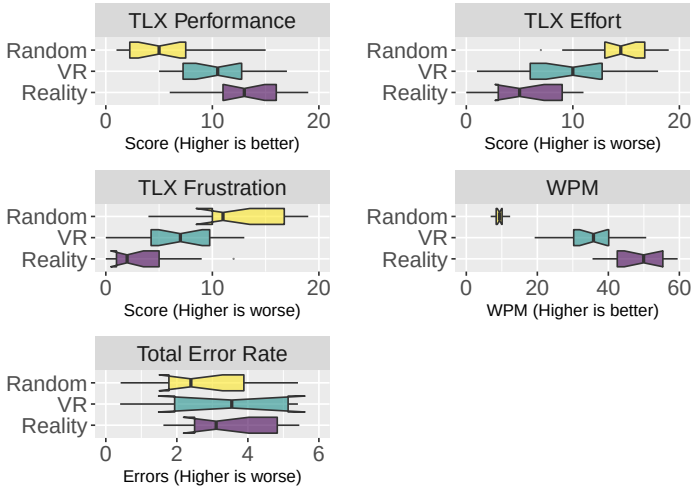


Fig. 4. Plots of measures for baseline typing performance.

Typing on the real keyboard with a monitor in *Reality* featured the lowest frustration and effort and highest Performance/WPM. However, the *VR* condition featured a comparable level of performance to previous studies, with an average WPM of 43.1, decreasing from 56.1 for *Reality*, with significant but small increase in error rate. This is inline with other tracked keyboard and fingertip implementations (e.g. [105] had a mean WPM of 41.2, [11] 53 WPM with training using the Logitech Bridge SDK, [48] and [37] in the mid-to-high 30s, [90] at 21 WPM),

	RM ANOVA				*Post hocs*		
Measure	F(2,34)	p	η_g^2	BF^{10}	Reality-VR	Reality-Random	VR-Random
WPM	136.42	<0.01	0.62	>150	t=5.5, p<0.01	t=16.2, p<0.01	t=10.7, p<0.01
Error Rate	7.1	<0.01	0.19	27.9	t=-2.9, p=0.02	t=0.5, p=0.9	t=3.5, p<0.01
TLX Performance	21.1	<0.01	0.43	>150	t=-2.7, p=0.02	t=6.5, p<0.01	t=3.8, p<0.01
TLX Effort	24.5	<0.01	0.48	>150	t=-2.4, p=0.05	t=-6.7, p<0.01	t=-4.2, p<0.01
TLX Frustration	27.3	<0.01	0.5	>150	t=-2.6, p=0.04	t=-7.0, p<0.01	t=-4.4, p<0.01

Table 1. Quantitative results for baseline typing performance.

albeit such comparisons should be treated tentatively as a ballpark guide, given the influence of parameters such as the physical keyboard design or inter-subject performance differences. These results suggest that our VR keyboard implementation was at least broadly comparable with the state-of-the-art in research with respect to tracked keyboard and fingertip typing performance, and assessed alone was performant enough to use for the subsequent evaluations. Examining performance in the *Random* condition, we find a mean WPM of 9.3, well in excess of the 3.8 WPM in [60] as previously discussed for secure keyboard implementations, and in excess of the mean entry rate found by Schneider *et al.* [90] of 6.57 WPM for their localized `RegionShuffle` condition, and 6.03 WPM for their `RowShuffle` condition. Indeed, the discrepancies emphasise the importance of the underlying keyboard and finger tracking when contextualising performance in different use cases - for which even our implementation is still some way off baseline performance, a point to be considered as a caveat for all our subsequent findings. Finally, the error rate was lowest in *Random* - this suggests that a high degree of accuracy could be achieved in singular key presses.

3.4 Part 2: Shortcut Discoverability

To assess shortcut discoverability, we created keyboard mappings for Microsoft Word (Office 365 2018), where icons were rendered on their associated keys based on the current state (e.g. if `CTRL` was pressed, then keys would show an icon reflecting their modified action, such as S showing the save icon for `CTRL+S`) For ecological validity, we used icons taken directly from Microsoft Word for these visualizations, extracted using ImageMSO [51] with icons selected based on keywords matching the shortcut action. Our participants reported using computers for 47±19 hours a week on average, of which approximately 5±5 hours was usage of Microsoft Word. Participants rated their level of expertise (1 being novice, 10 being expert) as 5.6±1.1 on average.

Fig. 5. VR Keyboard layouts for the **Icon** condition with (top) non-modified F-key shortcuts being visualised and (bottom) CTRL-modified shortcuts being visualised when the CTRL key was pressed.

Our intention was to compare shortcut recall both with and without the presence of a visual Icon, defining two conditions: **No Icon** and **Icon**. For both conditions, participants were prompted with a given action (e.g. "Save") and asked to enact the equivalent keyboard shortcut (e.g. *CTRL-S*), if known. If it was not known, participants could hit a "don't know" button on the keyboard, at which point the task would skip to the next action. This skip would also happen automatically after 8 seconds. Responses were recorded for analysis. We chose to evaluate 32 actions in a randomized order, covering a small subset of keyboard shortcuts available in Microsoft Word, selecting ones for which there was an image counterpart found in ImageMSO.

Performance metrics (identification accuracy (%), questionnaires (TLX workload, System Usability Scale [13]) and preferences both between conditions, and the preferred means of enacting a given action (either *no icon* meaning keyboard shortcuts without visual assistance, *Icon* meaning keyboard shortcuts with visual assistance, *GUI* meaning mouse/touch interactions with GUI elements, or *no preference*) were recorded. We deliberately did not compare directly to use of the Office ribbon GUI toolbar: given users had familiarity and exposure to this, we did not want them to judge the GUI on the basis of the VR view (lower fidelity due to headset) which might

unduly lead users to select the MR keyboard conditions, but rather judge based on their existing preferences. In addition, three questions were asked regarding shortcuts being error prone, easy to learn and efficient to use, based on questions from [89].

3.5 Results

	Accuracy (%)			User Preferences (%)			
Action	No Icon	Icon	Δ	No Icon	Icon	GUI	None
Save	89	83	-6	**89**	11	0	0
Cut	56	94	+38	**78**	17	0	6
Copy	89	100	+11	**94**	6	0	0
Paste	83	100	+17	**94**	6	0	0
Select All	72	67	-5	**78**	17	6	0
Bold	83	83	0	**61**	17	17	6
Italic	78	89	+11	**61**	17	17	6
Underline	83	89	+6	**61**	17	17	6
Decrease font	0	61	+61	6	**67**	22	6
Increase font	0	72	+72	11	**56**	28	6
Cancel	11	61	+50	44	44	11	6
Undo	78	83	+5	**56**	39	6	0
Redo	39	78	+39	33	**61**	6	0
Print	78	100	+22	**61**	11	22	6
Search	61	61	0	**50**	33	17	0
Replace	0	28	+28	22	28	22	28
Go to page	6	11	+5	6	44	39	11
Font dialog	0	39	+39	11	22	**56**	11
Align center	6	33	+27	17	33	44	6
Align left	11	50	+39	17	39	39	6
Indent left	0	33	+33	11	**56**	28	6
End of doc.	11	28	+17	22	**50**	22	6
Beginning	11	28	+17	22	**50**	22	0
1 word left	22	33	+11	28	39	28	6
1 word right	22	39	+17	28	39	28	6
Help	6	94	+88	11	**61**	17	11
Repeat	0	61	+61	22	**61**	17	0
Spelling	11	100	+89	17	44	33	6
Update fields	0	39	+39	0	39	44	17
Keytips	0	39	+39	11	**61**	6	22
Save as	0	56	+56	28	**50**	22	0

Table 2. Shortcut statistics broken down by action, showing the identification rate and the user preferences. Green highlighting refers to an improved identification rate, whilst grey highlighting noted that modality was preferred by the majority (over 50%) of participants.

As can be seen in Table 2 and Figure 6, visual assistance helped in discovering shortcuts, with some shortcut discovery rates improving by as much as 89%. On average, shortcut identification improved from 33% in *No Icon* to 63% in *Icon* (see Table 3) over the 32 tested shortcut actions. There were no significant effects on workload, nor

the duration taken to select the shortcut for the correct responses. In our questionaire deployed at the end of the study, in a choice between Icon or No Icon, the *Icon* condition was preferred by the majority of users, seen as less error prone and easier to learn.

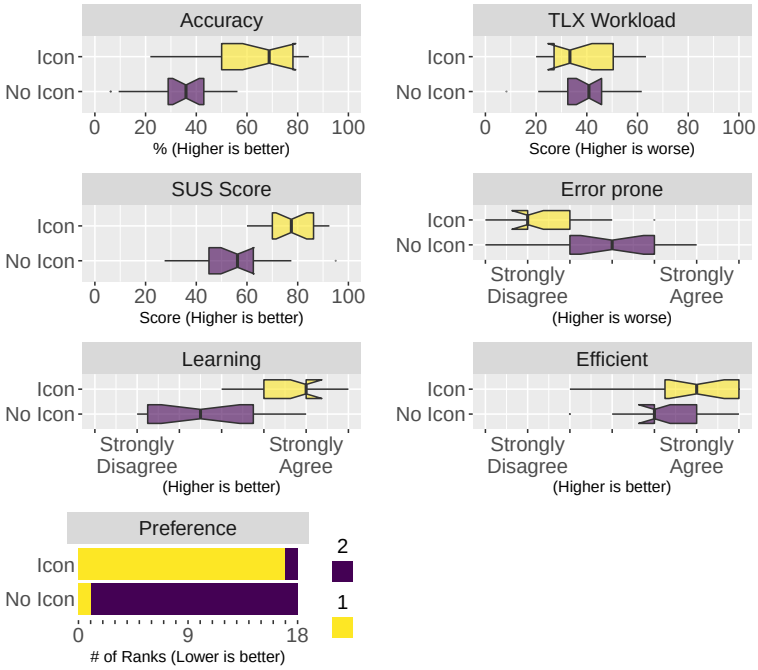


Fig. 6. Plots of measures for shortcut discoverability.

	RM ANOVA				95% CI	
Measure	F(1, 17)	p	η_g^2	BF^{10}	*No Icon*	*Icon*
Accuracy	87.3	<0.01	0.48	>150	[25.7, 41.3]	[55.4, 71.0]
Duration	1.46	0.24	.02	0.6	[2.2, 3.1]	[2.5, 3.3]
Workload	0.01	0.91	-	0.33	[31.5, 45.1]	[31.9, 45.5]
SUS	27.6	0.01	0.4	>150	[49.9, 63.0]	[71.5, 84.6]
Preference	128	<0.01	0.79	>150	[1.8, 2.0]	[0.9, 1.2]
Error prone	12.4	<0.01	0.23	34.0	[2.1, 3.3]	[0.8, 2.0]
Learning	29.2	<0.01	0.47	>150	[2.1, 3.2]	[4.2, 5.3]
Efficient	4.2	0.06	-	1.8	[3.7, 4.8]	[4.4, 5.5]

Table 3. Quantitative results for shortcut discoverability. Duration refers to mean time to shortcut selection for all correct responses. Error prone: "These shortcuts were error prone"; Learning: "These shortcuts are easy to learn"; Efficient: "These shortcuts are efficient to use".

However, shortcut visualization was not universally useful. Examining preferences per-shortcut, we note a pattern: for well-known shortcuts the preference was for *No Icon*. For two particularly well-known shortcuts

(*Save* and *Select All*), having an icon actually fractionally decreased correct identification. There was one shortcut action,"Font Dialog", that was preferred as being enacted via the GUI. In a subsequent interview regarding these preferences, five participants noted the confusion that having icons for known shortcuts could cause, for example *"for the ones I knew I didn't need the visual aids and I think the visual aids were sometimes confusing"* (P1). Ambiguity between the Icon and the action led to some conflict: *"So F5 I know is refresh, but when I saw undo I just saw the arrow and I hit that (instead)"* (P10), and *"I found that when it showed a visual aid they actually slowed me down because I was looking for the icon even though I actually knew the shortcut"* (P15). One participant noted that the meaning of some icons was particularly difficult to interpret.

In contrast, the use of icons for unknown/new shortcuts was noted as helpful: *"I wasn't able to figure out all of the icons, but it definitely improved my performance just in understanding some of them... I think the visual aid for shortcuts was very useful"* (P15). Where a preference for GUI interaction was indicated for a given action, it was noted by ten participants that this was because a transition to GUI interaction (e.g. from keyboard to mouse/touch) was likely necessary after executing the shortcut action. For example, if the keyboard shortcut for font options is used, the user would likely prefer to interact with the font dialog with a mouse to manipulate the font style, or if the file open dialogue was instantiated, the mouse would likely be preferred for navigating the file system.

3.6 Discussion

3.6.1 Impact of Shortcut Visualization. Addressing RQ1, exposing shortcuts visually allowed users to find them within approximately 2.9s (compared to a mean of 2.7s without assistance). However, on average accuracy was only 63%. Thus, for the first time, we can properly place the effectiveness of envisaged keyboard shortcut visualisation into context. Despite using a common, well known application, iconographic shortcut visualization alone is not sufficient to make the breadth of shortcuts discoverable. Better icon design might however bridge this gap. Previously discussed techniques [1, 34, 62] could also be combined with visual keyboard icons with the aim of increasing recognition and discoverability. Interestingly, we also exposed a tension between offering visual assistance, versus potentially confusing users or slowing them down due to relying on visual search rather than muscle memory. Based on our findings, we suggest that when a shortcut is already known, a visual icon should not be provided. However, we can imagine edge cases where this rule might be relaxed. If the user is new, or has not used the application recently, then the gamut of shortcuts should be exposed, either all at once, or staggered in such a way as to not overload the user. In such a case, shortcuts might adaptively fade over time or usage. More broadly, our results emphasize that per-keycap visualization of shortcut icons is a powerful feature that can (if mis-used) decrease the usability of visualised actions.

3.6.2 Implications for Engineering Augmented Keyboards. Our findings emphasised both the utility of visualisation and the potential for visualisations to mislead or distract the user when seeking a particular keyboard shortcut. Both points have significant implications for how we might design an architecture that can enable per-application keyboard augmentation.

Firstly, our example looked at only one set of common Word shortcuts. However, application shortcuts will frequently change depending on the context/state of the application. Moreover, despite an application and its associated shortcuts being active, there will still exist other keyboard shortcuts and actions that may be active. For example, the operating system will have global actions bound to keys, whilst other applications may also contribute to the active keyboard shortcuts despite not being in focus. On reflection, this emphasised that hierarchical mappings and inheritance would be a necessary key feature to support augmented keyboard interactions that take into account how applications and the underlying operating system bind to keyboard inputs.

Secondly, the errors our users made suggest that a range of interventions could potentially better support the discoverability of shortcuts. We might, for example, control the visibility of a shortcut based on some parameter (e.g. demonstrated prior usage/knowledge). Alternatively, we might select different 2D or 3D visualisations to support users, for example, rendering additional text on or around the key when a query gesture such as a fingertip dwell above the key is performed. This emphasised that the display space above the keyboard was not static. To better support comprehension, keycap augmentations are needed to encapsulate logic regarding dynamic visualisation of a variety of 2D/3D elements.

The requirement to support these capabilities re-affirmed, in our view, the need for a toolkit that could support users in creating such behaviours and managing the display space on/above the keyboard.

4 DESIGNING THE KEYBOARD AUGMENTATION TOOLKIT

4.1 Motivation

Our initial study underlined the impact that keyboard augmentations could have on the usability of existing keyboard shortcuts, significantly improving discovery rates in existing infrequently used shortcuts. More broadly, our literature review demonstrated the breadth of interactions envisaged for physical, augmented and virtual keyboards. Problematically however, we also noted that a significant proportion of this research into keyboard interaction was closed source - hindering replication and consequently making it difficult to compare past interactions to new interactions (without requiring a ground-up re-implementation) or extend past interactions. This is despite there being a number of commonalities across keyboard research, in terms of:

Key Interactions Appropriating sensed inputs on/around the key to provide additional states beyond binary keypress. For example, [10, 32] suggested touch interactions on keycap surfaces; [100, 113] explored the mid-air / above-keyboard space; [80] explored touch-sensitive keyboard states.

Key Display How the space on/around keys is visually augmented e.g. to indicate the presence of a shortcut through text, emojis, icons etc., or support feedback regarding on/around key interactions. Past research has predominantly focused on 2D per-key displays, however recently [90] demonstrated the potential utility of displays spanning multi-key elements, and 3D element above keys as part of the extended augmented keyboard space.

Context-Dependent Interactions (Keyboard Mappings) Activating groups of key interactions and display elements based on the user's current context e.g. based on the current application being used. For example, [10, 32] referred to context-dependent mappings and managing multiple mappings; [90] demonstrated multiple mappings.

Keyboard Layout Manipulating the spatial arrangement and dimensions of the physical or virtual keyboard and it's keys e.g. to better support mid-air text entry. For example Yanagihara [111] explored spherical/3D keyboard configurations; more typically 2D representations are standard, such as those explored by [21] in examining the impact of key shape and dimensions.

Whilst prior research has explored individual aspects of these features, there is a research bottleneck in terms of building novel combinations and permutations of these features that are integrative of prior research. For example, how might a multi-key interaction work with a 3D virtual keyboard? Or how might GestAKey inputs be bound to 3D above-key display elements? With every keyboard interaction, layout and mapping being implemented from the ground up by researchers, such integrative extensions of research become ever more difficult to achieve.

Running parallel to our research was the development of the Logitech Mixed Reality Keyboard, a prototype consumer-oriented keyboard based on the Logitech k780 whose position could be tracked by Microsoft Mixed Reality headsets.

This was the descendant of the Logitech Bridge SDK, in turn influenced by our work in bringing physical keyboards into VR using pass-through cameras [64]. The internal development of the Bridge SDK and this

Fig. 7. *Left:* The *Logitech MR Keyboard* for Microsoft Mixed Reality, in reality. See Figure 1 for an equivalent image rendered in VR. *Right:* Logitech test lab with a user interacting with the Logitech MR Keyboard.

peripheral kickstarted, and emphasized the need for, the concurrent development of KAT to support per-key augmentations. The Logitech MR keyboard utilizes the inside-out tracking on Microsoft MR headsets to both track a known active IR constellation (to allow for positional tracking), and capture/render a thresholded and segmented video pass-through of the user's hands within the boundaries of the keyboard, at low latency. This latter feature is crucial in particular for supporting performant text entry in VR, as it removes the additional latency and potential inaccuracy of current hand tracking implementations. In internal benchmarks, this prototype has demonstrated performance at approximately 85% of real world WPM (baseline mean of 50.6 WPM ±10.16 versus a VR mean of 43.4 WPM ±17.37). Considered in combination with advances in headset sensing such as the Oculus Quest hand tracking[1], we could see that our capacity to track our keyboards, and detect interactions on/around said keyboard, would increase greatly in the coming years. For example, hand tracking suggested that flexible per-key states/events were a necessity, such that keys might support pressed/touched/hover states, or react differently based on the finger interacting with the key. This was backed up by recent research into touch-sensitive MR keyboards in particular [80]. Prior novel hardware, such as GestAKey [93] could, for example, implement per-key pressure states in such a way. These developments provided further motivation for KAT, and influenced the identified requirements.

4.2 Requirements

Our proposed solution to the challenge of facilitating the replication, sharing and prototyping of phyiscal, augmented, and virtual keyboard interactions is the Keyboard Augmentation Toolkit (KAT), currently at a "beta" standard[2]. Based on the identified commonalities in prior research, and heavily influenced by the featured and requirements of leading augmented keyboard visions such as TDKs [10, 32] and ReconViguRation [90] and our evolving capability to track physical keyboards and hands, we have built KAT to provide a single Unity-based toolkit for creating novel virtual, augmented and physical keyboard interactions. The aims of KAT were to:

- Support notable, key features from prior literature in generic, re-usable ways
- Support extension of these features, facilitating future research
- Demonstrate an architecture that could inform future platform implementations.
- Where possible, re-use existing established external tools (e.g. for specifying mappings and layouts) better facilitating prototyping.

[1] oculus.com/blog/oculus-connect-6-introducing-hand-tracking-on-oculus-quest-facebook-horizon-and-more/?locale=en_GB
[2] See github.com/mark-mcg/keyboard_augmentation_toolkit for the latest release of KAT.

- Unify research into virtual/mid-air and augmented/physical keyboard interactions into a common shared platform.

With particular reference to the key features identified across physical and virtual keyboard research, KAT's requirements were to support:

Extensible Key Interactions From capacitive sensing, to raycast controllers, to hand tracking, the space on/around a (physical or virtual) key is now able to be appropriated for interaction in a variety of ways, with research continually exploring novel sensing and input modalities. Consequently, the events that could emanate from a key (or collection of keys) need to be extensible, enabling practitioners to define new interactions that can be replicated and shared.

Key Event Interception, Suppression, Simulation Where possible, support the interception, suppression and simulation of key events, allowing practitioners to re-write existing physical keyboard functionality based on proposed mappings and simulate physical keyboard outputs with virtual keyboards.

2/3D and Multi-Key Displays The space on/above/around a keycap could be appropriated for interaction and display, and this space may vary based on the keyboard layout and the current dimensions of a given key, or indeed span multiple keys.

Hierarchical Keyboard Mappings Applications could expose multiple keyboard mappings depending on their current focus, whilst users might have their own preferences (e.g. binding shortcuts to the number pad). Consequently, we need to be able to support the definition and usage of different keyboard mappings, inheritance in keyboard mappings, and different behaviours in conflict management where multiple mappings attempt to utilize the same key location.

Changeable Keyboard Layouts We need to be able to support the import or definition of 2/3D virtual layouts such that novel layouts are shareable. The virtual keyboard layout might vary over time, adapting to mobility (e.g. mid-air input), context (e.g. sitting down at desk, aligning to existing surfaces), or available input modalities (e.g. adapting to an existing physical keyboard to create an augmented keyboard). Where possible, interactions and per/multi-key display elements should be able to handle this plasticity in keyboard layout.

Our aim for KAT was to support generic extensible events, existing keyboard mappings, and importing/defining 2/3D layouts that could be changed in real-time, providing a powerful platform for prototyping of novel augmented and mid-air keyboard interactions. In this section, we describe the structure of the toolkit at the time of writing, and elaborate on the supported features and current notable limitations. It should however be noted that the version described in this paper is a snapshot, and consequently is liable to change/improve in the coming months/years. In particular, limitations described may not be present at the time of reading.

4.3 Architecture

A high level overview of the architecture of KAT can be seen in Figure 8. KAT's architecture is unique in that it facilitates the separate definition of:

Virtual Keyboard Layout The 2/3D design of the virtual keyboard (assumed to be one mesh per key), which can optionally match an underlying tracked physical keyboard to facilitate augmented physical keyboards.

Keyboard Mapping The interactive augmentations to be applied to the current keyboard layout. Each mapping comprises of one or more *KeyElements*.

For reference in the following discussion, we also refer to the following throughout:

Key Element Defines a unique interaction in a keyboard mapping, associated with one or more *KeyLocations*. These Elements can react to extensible sensed events (e.g. key depression, touch, hover) and can display UI elements that are sized to fit the 3D bounds of the associated physical-virtual keys. For example, an

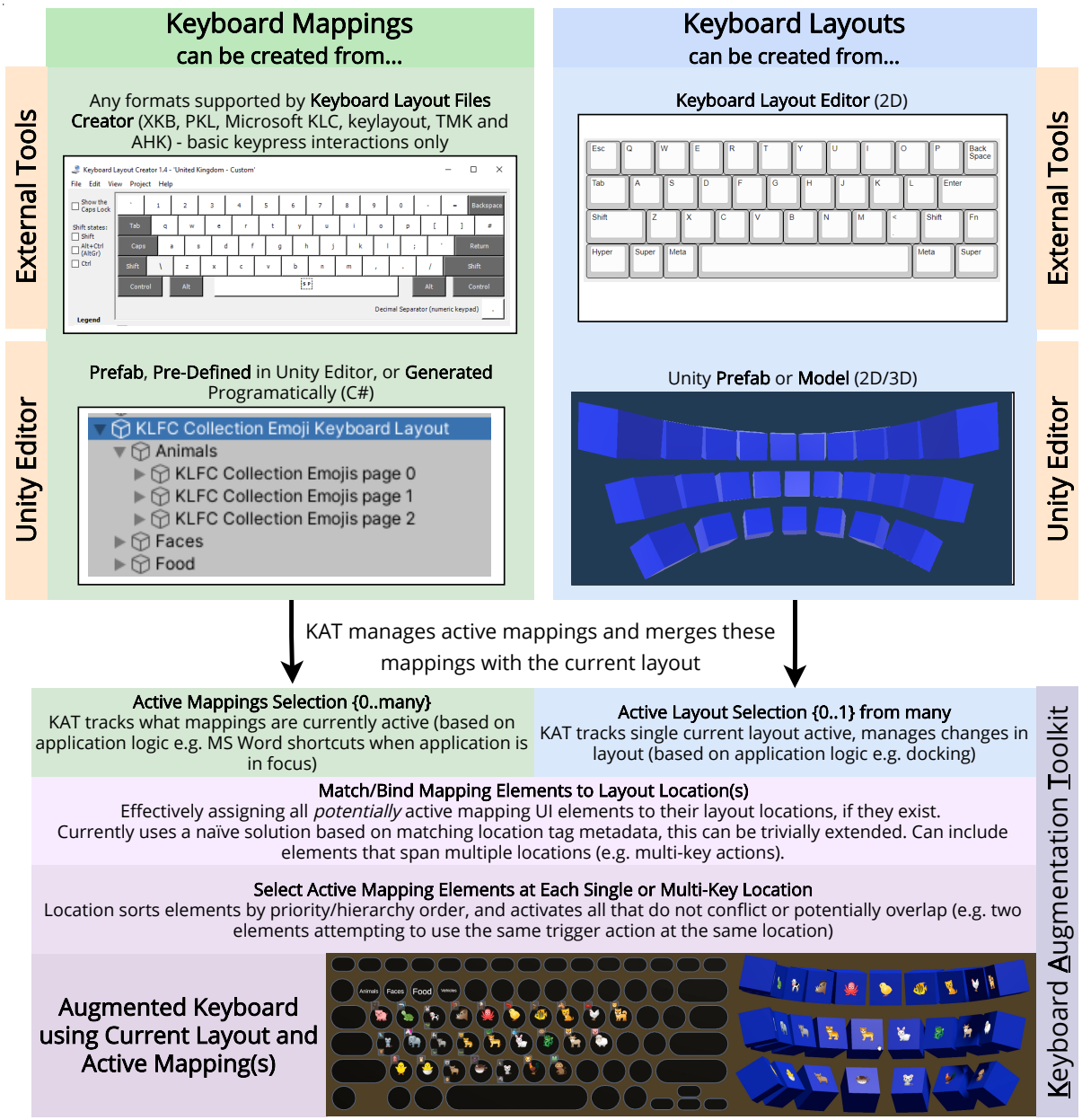


Fig. 8. An overview of the architecture of KAT.

Emoji mapping could have an element bound to the 'A' *KeyLocation* which renders a 3D smile model in the space above the key, and outputs a :smile: emoji when the 'A' key depression event is enacted.

Key Location A layout-independent definition of a key (described by keycode metadata such as the primary key(s) and modifier(s)) that provides *KeyElements* with access to the current underlying mesh/colliders based on the applied layout, and manages what assigned *KeyElements* are currently considered active (in the case of conflicts or multiple interactions being bound to a key). This acts as a bridge between our mapping and our layout.

Both the layout and current mapping can be dynamically changed, with KAT using *KeyLocations* to act as a bridge between the mapping and layout. The layout of a KAT keyboard can be changed on-the-fly. For example, consider that the layout might change to one preferable for mid-air direct touch input when the user is standing [22] and then change to an aligned digital twin of a physical tracked keyboard when seated to enable more performant input with a physical keyboard [64] - effectively enabling adaptation to the user's physical context. In the same way, we can also change the current mappings being applied to the KAT keyboard e.g. applying a unique keyboard mapping based on the current application in focus.

KAT manages how the layout and mapping(s) are merged to create virtual (or virtual-physical) keyboards with significantly expanded capability for interaction and display. It does so through decoupling the virtual key definition (i.e. the 3D mesh of the 'A' key for the given layout) from the persistent definition of that key based on metadata (i.e. the *KeyLocation* for the 'A' key) which in turn keeps a track of the current mapping elements suggested to be bound/active on that key (i.e. our emoji rendered on the 'A' key that is to be output when the key is pressed). In this way, we can change the mapping and layout separately, and the *KeyLocation* remains the same throughout. This flexibility enables practitioners to explore a variety of keyboard concepts, such as physically augmented and mid-air/surface-based keyboards, in terms of both their layout, and how we appropriate keys to enable new interaction capabilities.

4.4 Supported Usage

KAT provides a set of tools (predominantly Unity components with some software APIs) for creating novel interactive mappings combining different kinds of display (e.g. on or above key) and extensible triggers based not just on key depression events, but on any arbitrary event we can sense on or around a key e.g. triggering events when touching, tapping or hovering on a given key. Keyboard mappings and layouts can be created in the Unity editor using the provided C# components, or can be imported from supported third party tools. We do however note that KAT is not yet sufficiently mature for XR novices, and is currently best regarded as a toolkit for facilitating the prototyping of XR keyboards by practitioners and experts - some knowledge of how to use the Unity inspector is required, and there are notable caveats regarding usage (see subsection 4.7) e.g. it is not yet possible to import sophisticated mappings (e.g. using hierarchies, with multiple per-key interactions) from external sources.

4.5 Key Features

4.5.1 Key Capture & Representation (see Table 4). At an OS level, key inputs are captured as scan codes and then translated to platform-specific keycodes based on the keyboard driver. In turn, these key codes are typically interpreted by the platform's current active layout (a mapping in the parlance of this paper) which in turn may deliver Unicode key outputs for different languages, as well as key code events, to applications.

Problematically, these key codes are not entirely cross-platform compatible, and the underlying key code definitions are typically anglo-centric (e.g. Windows Virtual Keycodes), with other languages mapped onto the underlying keycodes. Whilst this issues persists with KAT, we do employ a platform-independent keycode representation, an extension of the representation used in Keyboard Layout Files Creator [40], with bindings

to translate between Windows, Mac, Linux and Unity key code representations. However, key interception, suppression and simulation is implemented for Windows only currently, a limitation we believe is acceptable for the time being as Windows is the only platform capable of driving the majority of tethered PC VR headsets currently. Consequently, KAT can support the mirroring of Windows applications in VR (using any appropriate screen mirroring library such as VR Desktop Mirror [17], with interception/suppression/simulation of key inputs to those applications.

Feature	KAT	Support Notes & Limitations
Key Capture & Representation		
Platform-Independent Key Event Representation		Adopted modified version of [40] platform independent key position with conversions for Windows, Mac, Linux, and Unity KeyCode.
Key Interception / Suppression		Using the `LowLevelKeyboardProc` hook Windows API, necessary to create Unity apps that can act as a proxy between the MR view and existing OS applications brought into Mixed Reality. Supports interception and suppression of key events (e.g. overriding a key press).

Table 4. Key Capture & Representation features and support.

4.5.2 Virtual Layout (see Table 5). Keyboard layouts are effectively comprised of a collection of meshes, where each mesh indicates a unique key position based on the representation above. These meshes can be built from 2D specifications, with basic support for importing from Keyboard Layout Editor [85]. However the primary means of defining these layouts is either through an imported 3D model that features appropriately named key submeshes (so Unity-specific key location meta data can be mapped to these submeshes), or has already been manually modified to incorporate key location meta data within Unity. Examples of all of these approaches can be seen in the associated video with this paper, notably:

From external 3D model We automatically extract the keycap surfaces from the Logitech MR Keyboard 3D model, creating a 2D keyboard overlay that perfectly fits the 3D model but can be dissociated from it.

From external 2D KLE definition We import (JSON) and instantiate the standard ISO 105 layout from KLE [85], constructing 2D keycap meshes.

2D from within Unity We recreate the Dudley keyboard layout from their 2019 ISMAR paper [22] on performance envelopes of virtual keyboards.

3D from within Unity We approximately recreated the Yanagihara 3D keyboard from VRST 2019 [111] using Unity cube primitives, applying key location metadata to each cube as appropriate.

Of particular note is our ability to dynamically instantiate and apply a new layout, and retain our binding of active keyboard mappings to layout element locations. As demonstrated in the associated video, this gives us the unique ability to modify on-the-fly which layout is utilized by the user for keyboard-type interactions. Our existing implementation demonstrates the utility of this within the context of mobility in typing. We envision that the user may transition from seated, desk-based interactions with a physical keyboard, toward mid-air interactions when standing/mobile (e.g. an architect leaving their desk to walk around a roomscale presentation of a building). In the video, we illustrate that the virtual keyboard can effectively "dock" with any available physical keyboard, at which point the virtual keyboard changes to a layout appropriate for the underlying tracked physical keyboard. Conversely, if we undock the keyboard, we are not constrained to the layout of the physical

Feature	Illustration	KAT	Support Notes & Limitations
Keyboard Layout			
Keyboard Layouts		✓	**For 2D**, basic support for importing Keyboard Layout Editor (KLE) JSON definitions [features like rotation of keycaps not currently supported]. **For 3D, and 2D keycap surfaces from 3D**, either imported 3D models or Unity Editor Prefabs can be used (given correct naming of sub-meshes or presence of location metadata), with 2D layouts able to be extracted from 3D keycap surfaces.
Dynamic Layout Switching (Docking)			Support for switching layout dynamically e.g. supporting docking/undocking a mid-air keyboard with a physical keyboard based on requirements or mobility of user.

Table 5. Keyboard layout features and support.

keyboard (which may not be appropriate for mid-air interactions), instead for example triggering a change to the Yanagihara keyboard for mid-air direct touch typing.

4.5.3 Keyboard Mappings (see Table 6). Keyboard mappings are collections of Key Elements, effectively describing a UI overlay that should be applied to the current layout.

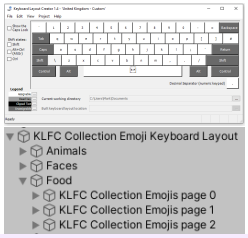
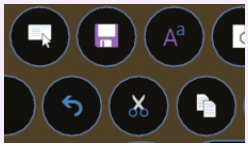
Feature	Illustration	KAT	Support Notes & Limitations
Keyboard Mappings			
Keyboard Mappings		✓	For **basic single-level mappings**, can import from any mapping compatible with Keyboard Layout Files Creator (top, e.g. Microsoft KLC) [No support for dead keys/chained dead keys and other advanced features currently]. **Hierarchical mappings**, and mappings with custom trigger actions beyond default selection, can be created (or composed from imported mappings) in Unity Editor (bottom).
Application Context Switches		✓	By polling the *ForegroundWindow* process name and window text, we can selectively activate context mappings based on what process is currently foregrounded (e.g. Microsoft Word, pictured), and what website is being browsed;

Table 6. Keyboard mappings features and support.

Basic single-level mappings can be imported from KLFC [40], a tool that supports the import and generic export of mappings from a variety of common platforms, giving us the ability to effectively import (JSON) from a

host of tools and existing mappings currently utilized by operating systems, for example from Microsoft Keyboard Layout Creator [70] or Linux XKB definitions, an example of which can be seen in subsubsection 4.6.1. This opens the door toward quickly importing or defining alternate languages, bespoke secure keyboard layouts etc.

Based on our examination of prior research, we identified that single-level keyboard mappings do not sufficiently allow us to share or manage the existence of shortcuts and actions from multiple applications simultaneously, nor do they allow for an application to adapt to it's own context e.g. taking over a selection of keys temporarily for an interactive widget during a transient interaction. Consequently, we also support hierarchical mappings. Hierarchical mappings allow for multiple mappings to be considered active at the same time. For example, consider a 'Photoshop' keyboard mapping that visualizes the default shortcuts of this application. Now consider custom shortcuts that might only be visible when the colour picker is active, and that override only a subset of the existing shortcuts. In KAT, we could create a colour picker mapping which extends the parent 'Photoshop' mapping, and then select (based on depth order within the hierarchy) which elements of the active mapping(s) should be active at each key location.

Our current implementation of this is naïve, effectively assigning all elements to their preferred key locations based on the target keycode metadata, and then letting each key location select the set of active elements based on hierarchy order and any possible conflicts (e.g. if two elements want to display their content on the center of the keycap surface) - but this approach is trivially extensible to support different assignment and activation behaviours. This approach allows an application to for example ask for a temporary widget mapping to become active, and because this widget is at the deepest point of the hierarchy, it gets priority over existing active mappings, such as the default language/text input mapping used by the user, or an extended parent mapping as in our example. We envisage this design is the first step toward a feasible OS/platform implementation of augmented keyboards that could operate correctly across multiple client applications, allowing for key inputs to be selectively and temporarily appropriated.

4.5.4 Interactive Key Elements. Each *KeyElement* in a mapping denotes a set of interactions to be bound to a set of one or more keys, handling both display (e.g. 2/3D icons, text) to support interaction discovery and feedback, and associated actions to be triggered based on received events.

Display (see Table 7. Each element can utilize the 3D volume and surfaces of the associated key(s), with exemplars (see Table 7) demonstrating 2D keycap and 3D above-key feedback. KAT provides helpers for querying the 3D bounds and surfaces of the associated key(s), through our definition of an *InteractionRegion* associated with a Key Location, which automatically adapts interaction colliders and layout bounds (and informs display elements) based on changes to the keyboard layout. Key elements, and their associated interaction regions, can also span multiple specified key locations, with KAT automatically creating a merged mesh and appropriate colliders for the combined multi-key location - however this support is currently basic, handling multiple keys correctly only when they are adjoining keys.

Interaction and Events (see Table 8). KAT supports extensible event generation and handling, crucially enabling the creation of novel key interactions that go beyond reacting to basic key depression events. As can be seen in Figure 9, events can be generated and relayed globally (e.g. on layout/mapping change) or routed to specific key locations based on applicability (e.g. relaying sensed data per-key such as pressure or touch).

Each key location can have a set of default interaction event providers, with KAT *InteractionRegions* by default attaching *CollisionEventProviders* for 3D colliders denoting the key mesh volume, the key surface, and the space above a key, triggered when a tracked fingertip enters these colliders. This forms the foundation of support for a variety of 3D interactions on/around physical-virtual keys. Each mapping key element can also attach additional interaction providers that will function whilst the parent mapping is active. For example, an element might attach

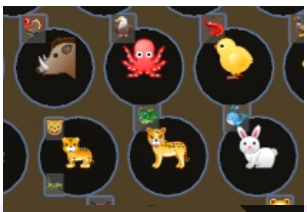
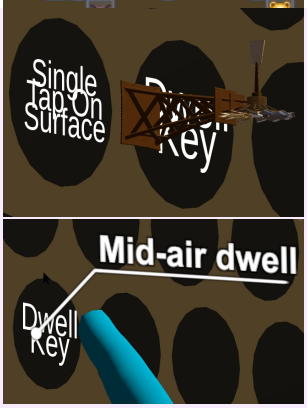
Feature	Illustration	KAT	Support Notes & Limitations
Key Elements / UI			
Key Location 2D Display		✓	*KeyLocation* display divided into regions (top/middle/bottom, left/center/right) with each region displaying Text or Sprites (using Unity TextMesh-Pro, with Unicode support), or optionally containing Unity native UI elements (e.g. sliders, buttons).
Key Location 3D Display		✓	*KeyLocations* have a 3D position and above-keycap bounding box. Consequently, we can use this space for feedback e.g. meshes/3D Models can be positioned within the bounding box above the keycap surface (here shown with animated windmill); and feedforward tooltips [102] can be drawn that call out from the center of the keycap.
Multi-Key Display		✓	Multiple co-located *KeyLocations* can be combined to form a composite location, complete with merged mesh, with all the same UI / interaction options available as per our 2D/3D display options, optionally to be triggered by pressing one or all of the keys included. These multi-key elements can also be scaled to fit the above-key space, enabling basic mid-air widgets above the key location (e.g. a mid-air slider, pictured bottom).

Table 7. Keyboard Elements / UI features and support.

a *TapEventProvider*, which listens for collision events from the virtual key surface collider and raises Tap events based on short repeated collisions.

Regardless, all raised interaction events for a given *KeyLocation* will eventually be relayed to all active *KeyElements* contained at that *KeyLocation*. *KeyElements* can then specify how they react to incoming events using *EventTriggers*, for example enacting actions based on a user depressing the key, touching the key surface, dwelling above the key surface, or repeatedly tapping the key - all supported by default[3] with KAT (see Table 8).

[3]Fingertip interactions are currently only implemented for the Oculus Quest/2, but support for other hand tracking solutions such as from Ultraleap could be easily added through adding fingertip colliders with the associated KAT components to tracked fingertips

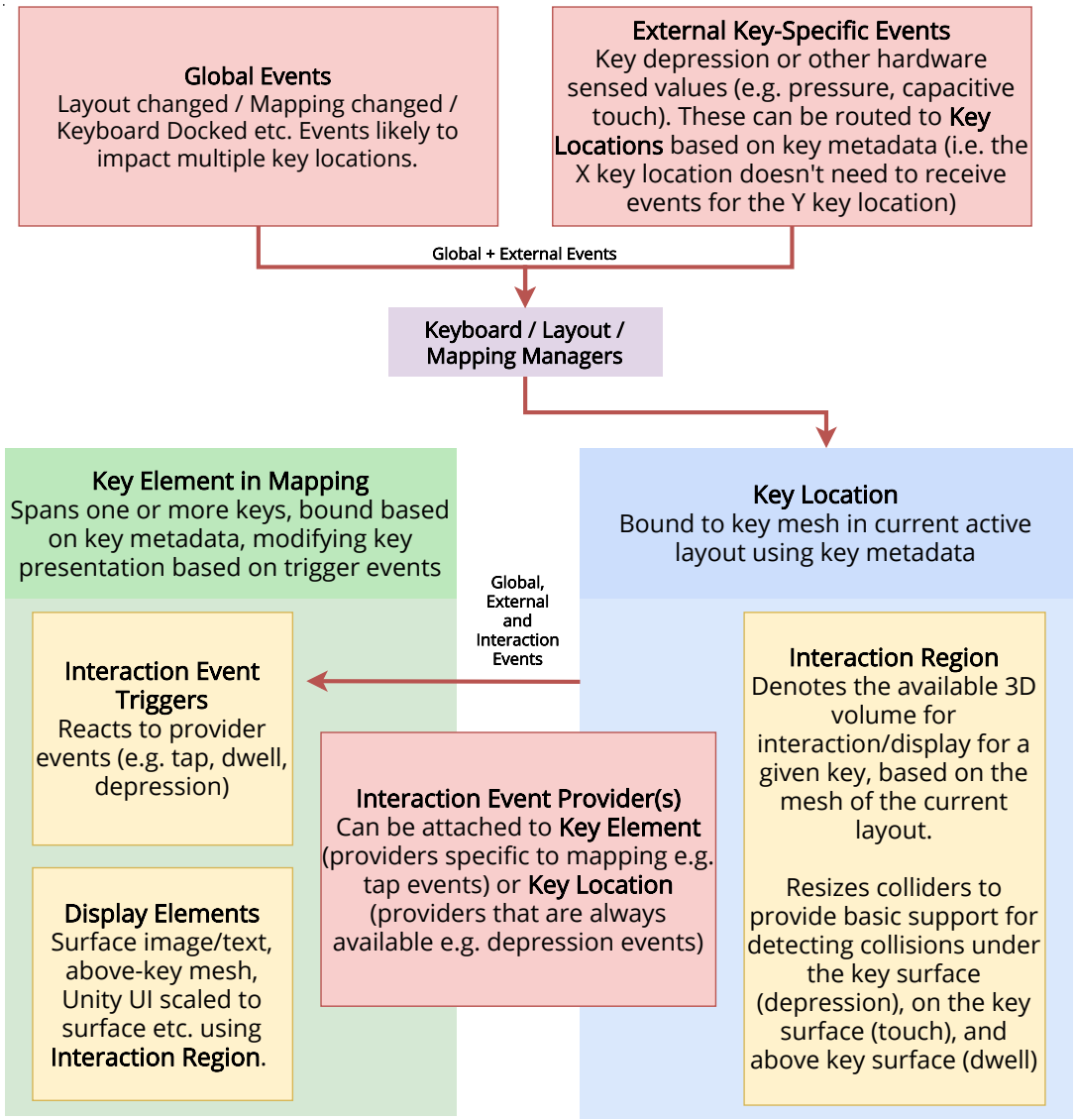


Fig. 9. Flow of generated events in KAT.

This functionality is trivially extensible, for example becoming feasible to support new interactions and sensing capabilities (e.g. pressure or tilt-sensitive keys) as we will demonstrate in subsubsection 4.6.4.

4.5.5 Dynamically Combining Mappings and Layouts. As discussed in the hierarchical mappings point, KAT provides the binding logic for connecting mappings with the current layout (illustrated in Figure 8), both of which can change over time. The assignment of mapping *KeyElements* to *KeyLocations* is handled by the *MappingToLocationBinder* which effectively matches the *KeyElement* location metadata with the available

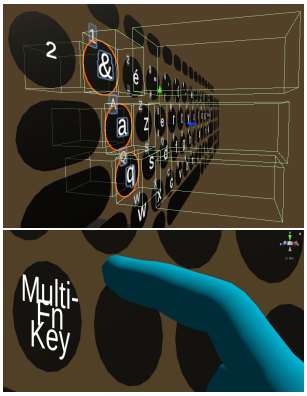
Feature	Illustration	KAT	Support Notes & Limitations
Key Elements / UI			
Element/Keycap Events & Interactions / Per-Key Touch-Sensing		✓	• React to interactions (Hover, Select, Surface Swipe/Tap, Dwell) driven by fingertip collisions (pictured). • React to platform events (e.g. key press) • Fake platform events when not docked with physical keyboard (e.g. enact key press on depression of virtual key) • Simulate platform key events (using *InputSimulator*, Windows only currently). • Extensible events and matching triggers, effectively accommodating a limitless range of key interactions. • N.B. Triggers cannot currently be defined/added to JSON mappings outside Unity Editor.

Table 8. Keyboard Elements / UI features and support.

KeyLocations based on the current layout. This is necessary as a number of keys are typically optional (e.g. number-pad keys) and may not be present in the current active layout.

Each *KeyLocation* manages which elements are active based on the behaviour of it's *ActiveElementSelector*. The default behaviour for this is currently to use depth in hierarchy, depth first, whilst avoiding obvious conflicts (e.g. two elements listening to the same key depression trigger). Locations are also responsible for relaying events to their active/contained elements e.g. a fingertip colliding with the location's key surface collider to indicate a tap interaction. For both assignment of *KeyElements* to *KeyLocations*, and the selection of currently active *KeyElements*, this logic can be modified through extensions to the *MappingToLocationBinder* and *ActiveElementSelector* classes.

4.6 Evaluation by Demonstration

To illustrate the capacity of KAT for facilitating novel virtual keyboard interactions, we discuss four evaluations by demonstration [53] of this toolkit, demonstrating how we re-create existing research mappings, layouts, and interactions, and create novel hierarchical mappings that expand the capability of virtual and augmented physical keyboards.

4.6.1 Existing Research Mappings. For keyboard mappings, we have provided some basic support for code-based generation of mappings, in particular generating randomized alphanumeric keyboard mappings, mimicking the secure keyboards presented by [90]. An example of keyboard mapping import functionality is shown in Figure 10, where new AZERTY layout (from [25–27], using the "afnor" XKB definition [29]) is used to generate a mapping. Note also this mapping does have some errors, particularly with missing unicode characters (indicated by the empty square symbol, font asset glyphs must be created in Unity for the target character ranges in TextMeshPro), and a lack of support for dead keys (keys which modify the next subsequent key press), and compose/keysym options currently. Nonetheless, this does illustrate that broadly functional mappings can rapidly be imported from existing sources, and we expect this support to be improved over time.

4.6.2 Hierarchical Mappings. An example of a hierarchical mapping can be seen in Figure 11. We define a parent mapping (either in the Unity inspector or programatically) which contains *KeyElements* that when activated load

Fig. 10. Example of the new AZERTY [25–27], imported as JSON generated by KLFC using the XKB definition [29], with alternate/shift keys also visualized.

a specified child mapping - here based on the emoji category selected, one of Animals, Faces, Food, or Vehicles. We then create child mappings of this parent set of controls, one each for the emoji categories. Each of these mappings in turn contains *KeyElements* that describe the emoji to be visualized (using a *TextMeshPro*-driven 2D display), and the action that should be executed on key press (outputting the emoji as text using a *StringOutputAction* which initiates a *PostMessageW* call on Windows to output unicode text to the current foreground application).

Fig. 11. Left: Unity component describing a child mapping with emoji *KeyElements* bound to each key. Right: "Animals" mapping has been activated, rendering available emoji shortcuts.

In the example figure, we have activated the "Animals" category of emojis by selecting the Animals key, and the numberpad controls for activating different emoji categories are still visible. This is because the Animals mapping is a child of the controls mapping, and thus inherits any (non-overloaded) augmentations for display. The KAT API supports combining or exclusively activating a given mapping, and KAT will attempt to activate and bind all elements in the child mapping to key locations, before doing the same for parent mappings. With this design, we can support nested hierarchies of mappings, for example enabling temporary widgets that are bound to a subset of keys but do not override other currently active shortcuts, or the intermixing of multiple active mappings simultaneously (e.g. base OS shortcuts, and specific shortcuts for the foreground application).

4.6.3 Existing Research Layouts. As part of our docking example illustrated in the accompanying video, we re-implemented recent VR keyboards from Dudley *et al.* [22] (2D) and Yanagihara *et al.* [111] (3D), pictured in Figure 12. These layouts both supported the full gamut of our toolkit interactions, and consequently are fully interactive after spending a matter of a few hours on their creation. In this way, we demonstrate in particular how novel layouts can quickly be prototyped and assessed by programatically or manually placing key location meshes into a Unity prefab, and make comparisons across virtual keyboard implementations easier.

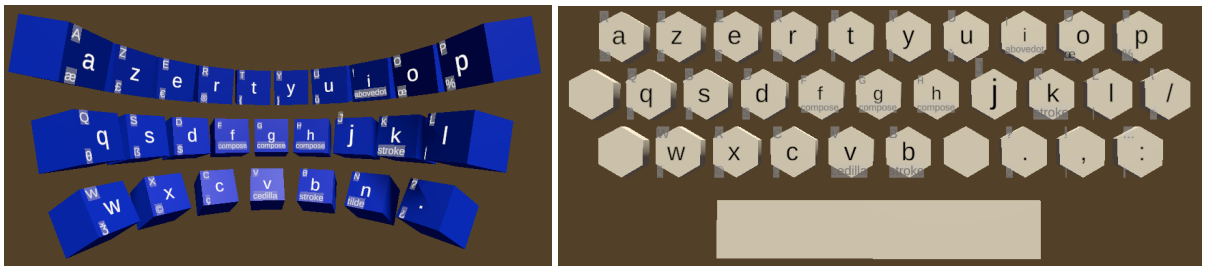


Fig. 12. Approximations of the Yanagihara *et al.* [111] (left) and Dudley *et al.* [22] (right) VR keyboard layouts, both using the previous new AZERTY mapping, with alt/shift key visualizations on keycaps.

4.6.4 Existing Research Interactions. Finally, to demonstrate how KAT can be extended to support novel key interactions, we implemented simplified simulated versions of GestAKey [93] - a key that could detect touch gestures on it's surface - and Metamorphe [6] - a key that, amongst other features, could be physically tilted left/right/up/down. To recap, for each interactive event type, we define an *EventProvider* responsible for raising the events as they occur, and an *EventTrigger* to enable key UI elements to listen and react to raised events.

Due to the achitecture of KAT, the implementation of these key interactions was trivial, as both could be implemented using existing fingertip-based collision events with our defined *InteractionRegion* colliders for each key mesh. In both cases, we defined a new *EventProvider* extending our base *CollisionBasedEventProvider* - a class that simplifies receiving collision events from a specific key collider (e.g. above key, on surface, or key body). This class drives a variety of our fingertip interaction event providers (e.g. for tap, touch and depression events). For our simulated Metamorphe key, we simply determined whether the fingertip collision was occurring to the right or left of the key. Based on this, we would modify the key mesh transform, rotating the key on the y-axis to simulate a left/right physical tilt on the virtual keyboard representation, and raise an associated *Metamorphe* event indicating the direction of tilt. A corresponding *MetamorpheEventTrigger* (extending our base *EventTrigger* type) could then receive this event and execute arbitrary *UnityEvents* (effectively invoking any function in Unity) based on this - in Figure 13 invoking *SetText* on the key element *TextDisplay* component to change the keycap text based on the tilt. Similarly, for the GestAKey implementation, when the fingertip collides with the keycap surface collider, this triggers continuous collision events that can be treated as coordinates on the 2D keycap surface - effectively turning the virtual keycap into a tracked touch input much as GestAKey functioned. Here, we use a basic swipe detection algorithm [50] to trigger swipe events (8 directions) on the surface, emulating a key function of GestAKey.

Both implementations took approximately 100 lines of code (excluding repetitive case statements) across three extension classes (the *EventProvider*, *EventTrigger*, and *Event*), with KAT providing the necessary scaffolding to quickly and easily integrate support for these additional keyboard interactions. In both cases, real hardware implementations would simply need a Unity class that could raise appropriate events scoped to the specific key they applied to (e.g. touch or gesture of the 'A' key). As detailed in Figure 9, these events could then be relayed

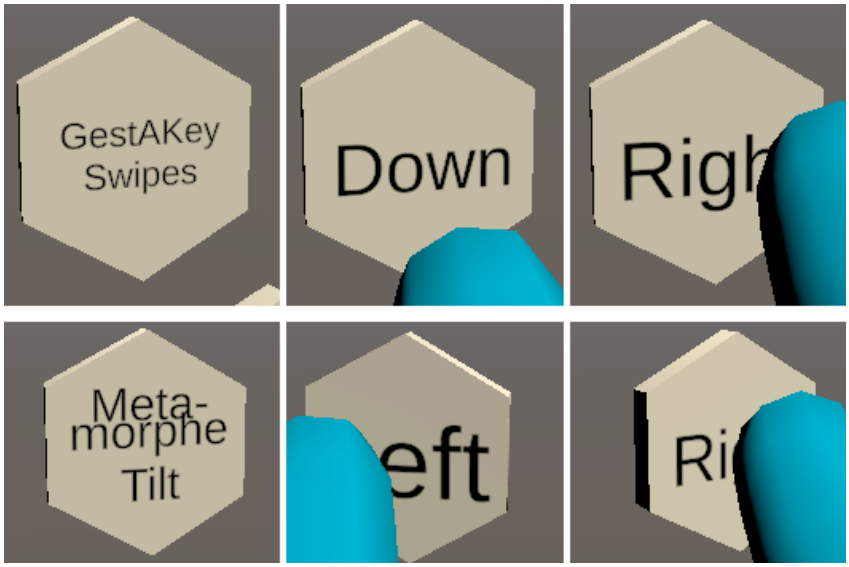


Fig. 13. Simulated GestAKey [93] (top) and Metamorphe [6] (bottom) in use. For GestAKey, we detect movements of the fingertip (here in blue) on the keycap surface and trigger associated swipe events. For Metamorphe, based on the fingertip colliding with the key mesh collider, the key is tilted left or right, with corresponding events generated that key elements can act upon.

to all relevant *KeyElements* to enact state changes (e.g. modifying the key transform to retain virtual-physical alignment) and trigger actions as appropriate. In this way, novel keyboard interactions based on sensed actions (e.g. hand tracking and collisions) or inputs (e.g. per-key pressure, tilt, capacitive touch etc.) become trivial to integrate into KAT, and consequently into mixed reality augmented keyboard usage.

4.7 Limitations, Omissions and Implications for Future Research for KAT

KAT features some notable limitations to be considered. Firstly, it is **tightly coupled to Unity**, with no cross-engine compatibility (e.g. for Unreal engine) designed for. Our intention is not that KAT will provide the basis for production-ready augmented keyboard implementations, or one platform that all XR experiences could currently utilise. Instead, KAT is intended to serve as a means of collaboratively scoping out the future of augmented keyboards in a way that is more compatible with open science/open source initiatives, allowing the sharing and re-implementation of different augmented keyboard presentations and interactions.

As a consequence, we have also prioritized a simplified design and choice of **naïve implementations (that are not production ready)** where possible, providing the most capacity for modification and extensibility at the cost of performance. For example, the number of draw calls required in the rendering of the keycap displays would be unthinkable for a production ready application, where KAT would end up using an unreasonable proportion of the rendering time. It should be noted however that for HCI-type evaluations, KAT is performant enough (i.e. VR frame rates will be maintained in any normal applications with no issue).

Regarding the creation of mappings and layouts, whilst we have some preliminary support on both counts, we do emphasize this support is nascent and developer focussed, with a **lack of mapping and layout designer tools**. Ideally we would see new tools developed that would make the design of mappings and layouts outside Unity easier, tools with in-built knowledge of the additional metadata required for augmented keyboards e.g. specifying new trigger types and events without having to implement them in Unity and add them using the

Unity Editor. We do intend to follow this path, and would encourage others interested in this to contact the authors directly for support.

Whilst we have implemented a host of interaction possibilities, many of these are **untested from a usability / user performance perspective** e.g. the hand-tracked tap/dwell/swipe of virtual keys. Whilst we intend to conduct evaluations of these features in time (post-Covid19), we emphasise that these implementations are provided as demonstrators of how we can implement events and triggers that extend the interaction capability of keys, be they purely virtual or physically aligned to a real key or surface. Implementations will need to be fine tuned depending on the use case and available sensing e.g. re-creating the specifics of mid-air touch handling for a complete recreation of the Yanagihara keyboard.

Indeed, the **available sensing** is a significant bottleneck in the creation and evaluation of these kinds of interactions. Variations in the robustness and accuracy of hand, surface, and keyboard tracking could significantly impact the usability of different designs. Our interaction implementations are demonstrated with the Oculus Quest hand tracking, but we readily admit that for some keycap finger interactions (e.g. swipes, taps) the accuracy of the tracking is likely insufficient to support these interactions entirely, given the fingertip can be multiple millimetres misaligned with reality, and actions have a significant latency (in the 10s of milliseconds by our estimation). The capability of the available sensing could introduce a significant confounding variable in assessing augmented keyboard interactions. Moreover, there is (as-yet) no one platform that supports *both* native hand tracking and a positionally tracked or located keyboard in a way that is accessible to developers. The closest that currently exists to this is the recently released Oculus Quest 2 support for optically tracking a Logitech K830 keyboard [77] as part of their push toward the "infinite office" [76] - however, developer access to this functionality is not yet available. Regardless, any research into augmented physical keyboards needs to carefully consider how best to unify accurate hand tracking with a locatable or trackable physical keyboard e.g. using Ultraleap with the Logitech MR keyboard, or using dedicated optical tracking such as Optitrack to enable both hand and keyboard tracking, as we used previously in this paper.

Reflecting on prior literature and existing UI frameworks, there are some **notable current feature omissions** to potentially be addressed in future versions:

Mouse interactions with keyboard Demonstrated in TDKs [10, 32], this feature does not make as much sense presently when considered against hand-tracked interactions

Spatial mapping strategies Also discussed in TDKs, this refers to the potential for placing keyboard shortcuts based on different priorities e.g. visual saliency, spatial constancy or recall. More recently, computational HCI approaches that optimize UIs could see significant utility if applied to the management of interactive key element assignment.

Keyboard widgets There is the potential for providing 1D interaction primitives for basic common interactions such as scalar controls that span multiple keys. Our toolkit makes creating these comparatively straightforward using the in-build Unity UI support, however we have not yet implemented concrete classes for primitives like this, and would suggest further development is necessary here.

Support for UI toolkits/frameworks Building on the widgets point, our platform provides a means of managing how UI elements are assigned and sized/positioned to virtual key locations, with generic UI display and event handling solutions, as well as support for using Unity UI elements as display elements. Support for other toolkits or frameworks, such as the Microsoft MRTK [71] is not yet present. However such support would be advantageous, opening the door to more sophisticated per-key interactions being supported by default. In time, if a defacto standard emerges regarding spatial UI definition and layouts for XR devices, we anticipate the approaches used in a mature version of KAT would guide how to design layout managers/constraints for UIs grounded on physical keyboards in particular.

And finally, we note that there is a wealth of research and extensions that will be better supported by KAT:

Keycap Icon Design: Static 2D icons could be expanded upon, for example using animation, 3D models, and off-key rendering such as floating tooltips, whilst previous research such as IconHK [34] could be applied directly to the keyboard. This is of particular importance given our findings regarding shortcut discoverability;

On/Around/Above Keyboard Widgets: New widgets that do not require a transition to an off-keyboard mouse/touch interaction could be explored. We could imagine hitting a colour picker key shortcut in Photoshop, which instantiates a mid-air picker within reach of the user's rested hand/finger, or spreads a colour picker over a multi-key area for recently used colors for example;

Different Contextual Modifiers: An understanding of efficient ways of allowing users to manage overloaded keyboard functionality, without undermining their mental model regarding what state the keyboard is in, will be crucial in encouraging adoption of new keyboard interactions, particularly taking into account the influence of mobility on layout, a feature we now support;

Keyboard Mapping Design: Whilst this has been extensively explored [10, 60, 90], we do not yet have concrete examples of successful mappings in-the-wild. Whilst we can envisage simple mappings with clear benefits (e.g. virtually enacting "the new AZERTY" [25, 27] as we demonstrate in subsubsection 4.6.1), our preliminary study into visualizing Microsoft Word shortcuts demonstrated that what may appear obviously beneficial might introduce unexpected pitfalls. Indeed, it is not yet clear precisely what *should* be visualized for different application contexts. Research has examined with considerable breadth the possibilities, without necessarily examining with sufficient depth the underlying effectiveness and utility in ecologically valid ways. What mappings might be envisaged for a browser or IDE, or an immersive gaming experience, and how best might we assess these in more ecologically and externally valid ways? Whilst we can support infinite hierarchies of mappings, to what extent can user's mental models actually navigate them? At what point do we go from effectively visualizing the chunking of multiple key presses [66], to overloading the keyboard and causing a breakdown in user comprehension? Moreover, support and tools will be required to define and visually or otherwise design hierarchical keyboard mappings and interactions agnostic of platform, in extensible, redistributable formats;

Augmented Keyboards in AR or Pass-through VR In principle, KAT would operate the same in Augmented Reality or Video See-Through (VST) / Pass-Through VR as demonstrated in VR-grounded MR throughout this paper. However, the requirements for keyboard/hand tracking in a pure AR experience would be far stricter - if off by even a few millimeters, the virtual key icons could appear misaligned with the key caps in reality, something not perceivable in VR. In addition, hand tracking would need to be accurate enough to handle occlusion of keycap renderings by hands/fingers. Such constraints need to be considered in the near-term, however they represent relatively minor impediments in our view.

The Relationship Between Physical-Virtual and Purely Virtual Keyboards With KATs ability to support dynamic, changing layouts, we do also open up something of a *Pandora's box* regarding how a user's mental model of a keyboard may need to adapt as we map interactions and UIs to physical keyboards, and then enable transitions toward mid-air or surface-aligned interactions as well. At what point should a mid-air interface abandon any pretences of being based on a keyboard layout, and does it make sense for a mid-air keyboard to mimic or provide augmentations designed for physical, tracked keyboards? We suggest there is a multitude of research to be conducted in exploring the differences between user expectations for augmentations when applied to tracked keyboards versus surface-aligned and mid-air keyboards, and the relationship between existing spatial / mid-air UIs and augmented keyboards.

5 GENERAL DISCUSSION

Previous research has expanded our capability to interact with keyboards [10, 60, 90] and learn available shortcuts [34, 61] through custom applications and hardware designs. Now, however, consumer MR headsets are improving

in both capability (resolution, sensing, etc.) and adoption, with further consumer-oriented headsets rumoured, particularly for AR [84]. Consequently, the relationship between these mixed reality headsets and the peripherals we interact with [64] should be re-assessed. Given the ability to render on, or around, any given peripheral in mixed reality, their form and function could be altered in ways that were previously impossible without expensive custom hardware solutions.

In support of this endeavour, we contribute KAT, a toolkit for prototyping novel augmented keyboard interactions and XR keyboard layouts. As we have discussed, KAT provides a framework within which new sensing, events, and per-key displays can be prototyped and applied to the current keyboard layout, which itself could be mapped to an underlying physical keyboard or surface, or presented mid-air. In this way, we provide practitioners with open source tools for **creating, and sharing, augmented and mid-air keyboard designs**, **prototyping augmentations of keyboards**, and exploring **new interactions on/around keys**. Through a combination of iteration and literature review, we have arrived at a core set of features needed to create a functional, re-usable augmented keyboard that can effectively support the visions of recent research. We have also demonstrated that the base capabilities of MR headsets (e.g. hand/finger tracking, head orientation tracking) can be exploited to introduce new possibilities in having the keyboard augmentations react to gaze (albeit further research is required here) and above/on-key events.

5.1 The Advantages of Augmented & Mixed Reality Keyboards

As this paper, as well as prior work on keyboard augmentations [10, 32, 60, 72, 90, 95], has reinforced, there are significant potential benefits to being able to visually augment a physical peripheral such as a keyboard e.g. in terms of:

Visibility of Shortcuts: Visualizing application shortcuts *in situ*, based on the current context;
Parity Between On-Screen Leyboards (OSKs) and physical keyboards: Dynamically altering keyboard mappings, mimicking OSKs in form and function;
Usable Security: Obfuscating the mapping between physical keypress and virtual action, preventing shoulder surfing;
Form Factors for Mobility: Actions could be dynamically allocated to keys, minimizing the number of physical keys necessary;
Above Device Interactions: The space above the keyboard could provide access to interactive widgets within reach of the hand, without a transition to other input device such as the mouse;
Expanded Key States: New key states can be defined and reacted to, based on which finger interacts with the key, whether the finger is hovering above or resting on the key, etc.
Common Sensing Platform: Interactions that were previously shown to be effective in research could be ported to a general purpose platform for the first time, without requiring bespoke hardware configurations.

It should be noted that not all of these points require the combination of a tracked or located keyboard and an MR headset. For example, visualizing shortcuts alone is possible with per-keycap displays such as on the Nemeio [72] or Optimus [98], and accordingly the benefits we demonstrated in our study would likely transpose to per-keycap displays. However, this combination does provide unique affordances: hand/finger tracking for enhanced key states; the ability to perform direct-touch mid-air inputs grounded/positioned based on the keyboard; rendering elements on/around the keycap with full 3D depth; offering the ability to transition between standing/roomscale interactions and seated interactions, mapping interfaces onto the physical keyboard to benefit from the haptics/proprioception it supports. In short, there are compelling arguments that this combination could see adoption in the future, particularly if virtual workspaces, driven by MR headsets, such as recently envisioned both in industry (e.g. the Oculus "infinite office" concept [76]) and in research [28, 65, 78] come to fruition.

However, there are alternate routes toward further enhancing our ability to interact using physical keyboards, of which *augmented keyboards* is just one such instance. Consider launchers such as Blur [18, 89], Alfred [87], Spotlight [3], LaunchBar [75], Wox [68] and Hain [54]. These allow users to type actions, commands and search phrases, whilst tools such as LaunchBar and the Windows Emoji shortcut [107] allow for special characters such as emojis to be inserted into a document with visual assistance (e.g. typing ':beer:'). Launchers provide a counterpoint to overloaded key actions [67] and the augmented keyboard vision of productivity. Instead of relying on spatial/contextual memory or visual search to recall a shortcut location and action, the user can state the intended action or outcome and find a best match. Our inclination is that these approaches could be complimentary: better launchers could be created, with frequently used actions mapped to keys and infrequently used actions found in a keyboard map or retrieved through launcher interaction. Augmented keyboards might even provide a pathway toward launcher adoption. However, research will be required to explore the tensions and overlaps between these two approaches.

5.2 Support for Physical and Virtual Keyboards on MR Platforms

There is, however, a significant roadblock to supporting augmented keyboards in MR more generally: the lack of support for physical keyboard text entry in MR platforms. As of today, the major consumer PC-based MR platforms have varying support for physical keyboards, often understandably focusing on controller-based text input using virtual keyboards or voice interaction. For example, on SteamVR, physical keyboard input is not even processed by the SteamVR platform web browser, chat or store applications. On SteamVR, it is up to the individual app developer to add keyboard support e.g. the popular Virtual Desktop app[4] *does* process physical keyboard text input, and in addition has powerful shortcuts to re-centre, change display for different types of content, etc. In contrast, the Microsoft Mixed Reality has more physical keyboard support. Most of the platform VR apps already process keyboard input, and there is a dedicated Windows hotkey for switching keyboard input focus between the traditional desktop and VR environments. Similarly, the Oculus platform supports bluetooth keyboards for text input. And the Khronos OpenXR standard provides methods to access peripheral state, including input and action bindings, but no support for keyboards [45].

Consequently, further support will be required for physical keyboard input in VR / MR platforms. Research can lead the way not just in imaginative interaction design possibilities, but also in specification and requirements, as well as providing evidence regarding utility and usability, and in answering core questions such as when to show/hide the keyboard; how to facilitate the specification and use of application/region keyboard layouts; how mappings should function across locales; what guidelines should govern the use of augmentation (which our paper in particular begins to address); and how best to support transitions between OSK/mid-air and physical keyboard use. If questions such as these can be addressed, then we can make a more informed case for tools and platforms to support such peripherals, particularly geared toward MR productivity use cases [65], in the future.

5.3 Mixed Reality Augmentations of Other Peripherals

The underlying assumption that we can track the users hands and gaze, and are wearing sensing capable of tracking peripherals and objects (either actively or passively) within our reach, lends itself to considering how we might best utilize MR to appropriate and augment these objects, of which the keyboard is just one example. There has been a multitude of research in this space, particularly in envisioning how appropriated objects and surfaces might be utilized to further productivity, for example:

Augmenting Peripherals Enhancing control surfaces [7] or enabling direct inputs on touchpads [31], or around-device interactions e.g. on mice [8];

[4] vrdesktop.net

Appropriating and Augmenting Objects Re-purposing existing objects for interaction [19, 33, 83, 106] and "annexing reality" [43] both generally and specifically in MR [5, 15, 112], with even the appearance of real-world objects being modified to convey feedback [57];

Appropriating Surfaces From existing tabletops and surfaces around keyboards [8, 42, 47, 74] to providing sparse haptic proxy surfaces [16], the space around peripherals has repeatedly been exploited.

This idea has been revisited time and again because there is a general sense that given sophisticated sensing and display technologies, we might better utilize available inputs and surfaces, and extract more performance from existing inputs and devices. The affordances of mixed reality do, however, place an additional emphasis on the use of depth and 3D rendering, and the exploitation of hand/object tracking, compared to much previous work which has been predominantly oriented on projecting AR interfaces onto physical surfaces. Indeed, with MR, every pressure sensor, capacitive surface, or button could optionally offer a greater capacity for interaction. Conversely, MR interfaces could benefit from being mapped to physical inputs such as the keyboard. Just as with augmented keyboards, we could imagine a standard mouse where we render multiple MR virtual buttons on the one physical button (e.g. clicking at the tip of the mouse versus the centre); or where the user could swipe a finger in any direction on the button to enact a mode switch; or even provide an additional floating mid-air button, exocentrically placed at an offset of the pose of the physical mouse button within reach of the user's finger, if they lift their finger off the button and reach forward. In such scenarios, the MR headset sensing is dictating the majority of the UI interaction, and the mouse button "click" event is effectively providing a certainty of input for a subtle haptic interaction, making up for limitations in headset-based hand and finger tracking. However, it is one challenge to imagine such interfaces, as HCI has done repeatedly over the past decades - it is another to practically support the creation of such UIs. KAT is a modest step in this direction, attempting to formalise *how* augmented and virtual keyboards might practically be designed for.

The broader challenge, however, is in supporting distributed, mixed reality spatial UIs and interactions that work across a multitude of peripherals, objects and surfaces. In the short term, the SteamVR Input Mappings [69] represent a scaffold upon which basic distributed interactions might be built, with all inputs having a prescribed pose (i.e. their position in real-world/virtual space) and exposing a variety of events that can be generated from inputs (eg. pressure mapping to a 1D variable or a boolean response). If more peripherals supported such a scheme, we might imagine that mice, touchpads, and other interactive peripherals or surfaces might be tracked and augmented dynamically in 3D space. In which case, a more generalisable toolkit might allow for attaching additional interface elements to the control surface, or the space around said surface, based on the capability exposed by the input modality. In the longer term, we might imagine that optimisation approaches could be employed to automatically distribute UI elements as appropriate [81], e.g. based on the available control surfaces, objects and peripherals at the user's desk. These are significant challenges not just in the design of interactive systems, but also in how they are engineered. Moreover, the benefits of such interactions, and evidence that they do not negatively impact workload and performance, will be required to justify potential adoption by application designers operating systems. This transition from 2D planar to distributed 2.5/3D UIs (i.e. flat UI elements placed in 3D space, or fully 3D UI elements with depth) offers the possibility of designing innovative new interactions, transposing research such as Magic desk [8] to MR media in the future.

5.4 KAT: A Toolkit Supporting MR Keyboard Research

Our aim is for KAT to provide the foundations for implementing and evaluating both new, and previously envisaged, augmented and mid-air keyboard interactions. KAT will assist in re-evaluating previous visions and research and defining a common baseline of functionality that might comprise the physical keyboard of the (near) future. For example, many of the features made possible on bespoke hardware and software implementations (shown in TDKs [10], ReconViguRation [90], and other systems based on gesturing or using additional key states

[14, 92, 93, 100, 113, 116]) could be built and deployed on such a platform. In making KAT available, we hope to provide the foundation for a variety of new research on *augmented and virtual keyboards*. We also encourage researchers and developers to contribute to KAT, along the numerous potential avenues we have detailed for improvement, enabling more and broader prototyping, sharing and collaboration of novel virtual / virtual-physical keyboard designs.

In time, we might expect that the features of KAT would be rolled into operating systems, with existing Keyboard input APIs modified to allow for querying physical key positions/dimensions/states, modifying the icon or model associated with a key, and displaying custom keyboard mappings per application. Such capabilities could equally apply to future keyboards with integrated per-key displays, if they end up as the evolutionary end-point of physical keyboards. Methods for managing what keyboard mappings are given priority, and how key-assignment conflicts are dealt with (e.g. balancing visual saliency and spatial congruence [24]) will need to be studied further. Usage and refinement of KAT by researchers and practitioners will help to scope out the requirements of such integration, and the HCI community can play a significant role in further establishing the utility of augmented keyboards, guiding research and practice regarding how best to utilise these new peripherals and capabilities.

6 CONCLUSIONS

This paper has discussed the development of the *Keyboard Augmentation Toolkit* (KAT), intended to support the creation of virtual and augmented keyboards for extended and mixed reality headsets. Using an initial prototype of KAT, we examined the impact and pitfalls of visualizing shortcuts on an augmented physical keyboard. Supported by this and other recent developments in augmented keyboard research, we then described the design, development and evaluation-by-demonstration of KAT. As an open source toolkit, KAT will better enable practitioners to prototype, create and replicate XR keyboard experiences, supporting the creation of enhanced per-key displays; flexible hierarchical keyboard mappings; enhanced interactivity and per-key states; and both 2D and 3D layouts that can be changed in real-time. KAT will enable practitioners to further push the boundaries of keyboard form and function using XR headsets, supporting research into virtual and augmented keyboard performance and usability. This will be crucial if peripherals such as the keyboard are to take advantage of future productivity environments that rely on spatial computing.

ACKNOWLEDGMENTS

Thanks to all at Logitech Design Lab in Cork, Ireland, and Logitech Lausanne, Switzerland, that helped in the formation of this paper and the ideas contained therein. This research was funded in-part by an EPSRC IAA project (EP/R511705/1, Novel Interactions for Mixed Reality). This research was also funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement #835197: ViAjeRo).

REFERENCES

[1] Jonathan Aceituno and Nicolas Roussel. 2014. The Hotkey Palette: Flexible Contextual Retrieval of Chosen Documents and Windows. *Proceedings of the 26th Conference on l'Interaction Homme-Machine - IHM '14*, 55–59. https://doi.org/10.1145/2670444.2670452

[2] Apple. 2018. Apple Touch Bar. https://developer.apple.com/macos/touch-bar/

[3] Apple. 2018. Use Spotlight on your Mac. https://support.apple.com/en-gb/HT204014

[4] Ahmed Sabbir Arif and Ali Mazalek. 2016. WebTEM: a Web application to record text entry metrics. *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces - ISS '16*. https://doi.org/10.1145/2992154.2996791

[5] Mahdi Azmandian, Mark Hancock, Hrvoje Benko, Eyal Ofek, and Andrew D. Wilson. 2016. Haptic Retargeting. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*. ACM Press, New York, New York, USA, 1968–1979. https://doi.org/10.1145/2858036.2858226

[6] Gilles Bailly, Thomas Pietrzak, Jonathan Deber, and Daniel J. Wigdor. 2013. Métamorphe: Augmenting Hotkey Usage with Actuated Keys. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 563–572. https://doi.org/10.1145/2470654.2470734

[7] Florent Berthaut and Alex Jones. 2016. ControllAR: Appropriation of Visual Feedback on Control Surfaces. In *ISS '16 Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*. https://doi.org/10.1145/2992154.2998580

[8] Xiaojun Bi, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2011. Magic desk: bringing multi-touch surfaces into desktop work. In *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*. 2511. https://doi.org/10.1145/1978942.1979309

[9] Mark Billinghurst, Adrian Clark, and Gun Lee. 2015. A Survey of Augmented Reality. *Foundations and Trends in Human-Computer Interaction* 8, 2-3 (2015), 73–272. https://doi.org/10.1561/1100000049

[10] Florian Block, Hans Gellersen, and Nicolas Villar. 2010. Touch-display keyboards: Transforming Keyboards into Interactive Surfaces. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, 1145. https://doi.org/10.1145/1753326.1753498

[11] Sidney Bovet, Noirin Curran, Aidan Kehoe, Mario Gutierrez, Thomas Rouvinez, and Katie Crowley. 2018. Using traditional keyboards in VR: SteamVR developer kit and pilot game user study. *The 2018 IEEE Games, Entertainment, Media Conference - GEM '18* August, 131–134.

[12] D. A. Bowman. 2020. Embracing Physical Keyboards for Virtual Reality. *Computer* 53, 09 (sep 2020), 9–10. https://doi.org/10.1109/MC.2020.3004605

[13] J Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* (1996).

[14] Daniel Buschek, Bianka Roppelt, and Florian Alt. 2018. Extending Keyboard Shortcuts with Arm and Wrist Rotation Gestures. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*, 1–12. https://doi.org/10.1145/3173574.3173595

[15] Lung-Pan Cheng, Li Chang, Sebastian Marwecki, and Patrick Baudisch. 2018. iTurk: Turning Passive Haptics into Active Haptics by Making Users Reconfigure Props in Virtual Reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press, New York, New York, USA, 1–10. https://doi.org/10.1145/3173574.3173663

[16] Lung Pan Cheng, Eyal Ofek, Christian Holz, Hrvoje Benko, and Andrew D. Wilson. 2017. Sparse haptic proxy: Touch feedback in virtual environments using a general passive prop. In *Conference on Human Factors in Computing Systems - Proceedings*, Vol. 2017-May. 3718–3728. https://doi.org/10.1145/3025453.3025753

[17] Clodo76. 2018. VR Desktop Mirror. https://github.com/Clodo76/vr-desktop-mirror

[18] Andy Cockburn, Carl Gutwin, Joey Scarr, and Sylvain Malacria. 2014. Supporting Novice to Expert Transitions in User Interfaces. *ACM Computing Surveys (CSUR)* 47, 2, Article 31, 36 pages. https://doi.org/10.1145/2659796

[19] Christian Corsten, Ignacio Avellino, Max Möllers, and Jan Borchers. 2013. Instant user interfaces: Repurposing everyday objects as input devices. In *ITS 2013 - Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*. 71–80. https://doi.org/10.1145/2512349.2512799

[20] Steve Draper. 2018. Effect size. (2018). http://www.psy.gla.ac.uk/~steve/best/effect.html

[21] Tafadzwa Joseph Dube and Ahmed Sabbir Arif. 2020. Impact of Key Shape and Dimension on Text Entry in Virtual Reality. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3334480.3382882

[22] J. Dudley, H. Benko, D. Wigdor, and P. O. Kristensson. 2019. Performance Envelopes of Virtual Keyboard Text Input Strategies in Virtual Reality. In *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 289–300. https://doi.org/10.1109/ISMAR.2019.00027

[23] Barrett Ens, Juan David Hincapié-Ramos, and Pourang Irani. 2014. Ethereal Planes: A Design Framework for 2D Information Space in 3D Mixed Reality Environments. *Proceedings of the 2Nd ACM Symposium on Spatial User Interaction*, 2–12. https://doi.org/10.1145/2659766.2659769

[24] Barrett Ens, Eyal Ofek, Neil Bruce, and Pourang Irani. 2015. Spatial Constancy of Surface-Embedded Layouts Across Multiple Environments. In *Proceedings of the 3rd ACM Symposium on Spatial User Interaction* (Los Angeles, California, USA) *(SUI '15)*. ACM, New York, NY, USA, 65–68. https://doi.org/10.1145/2788940.2788954

[25] Anna Maria Feit. 2018. *Assignment Problems for Optimizing Text Input*. G5 Artikkeliväitöskirja. http://urn.fi/URN:ISBN:978-952-60-8016-1

[26] Anna Maria Feit, Mathieu Nancel, Maximilian John, Andreas Karrenbauer, Daryl Weir, and Antti Oulasvirta. 2021. AZERTY AméLioré: Computational Design on a National Scale. *Commun. ACM* 64, 2 (Jan. 2021), 48–58. https://doi.org/10.1145/3382035

[27] Anna Maria Feit, Mathieu Nancel, Daryl Weir, Gilles Bailly, Maximilian John, Andreas Karrenbauer, and Antti Oulasvirta. 2018. Élaboration de la disposition AZERTY modernisée. (June 2018). https://hal.inria.fr/hal-01826476 working paper or preprint.

[28] Nadia Fereydooni and Bruce N. Walker. 2020. Virtual Reality as a Remote Workspace Platform: Opportunities and Challenges. (August 2020). https://www.microsoft.com/en-us/research/publication/virtual-reality-as-a-remote-workspace-platform-opportunities-and-challenges/

[29] Djyp Forest Fortin and Maxime Labelle. 2021. Implémentation de la norme AFNOR. https://github.com/Djyp/azerty_afnor

[30] Euan Freeman, Stephen Brewster, and Vuokko Lantz. 2016. Do That, There: An Interaction Technique for Addressing In-Air Gesture Systems. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*. https://doi.org/10.1145/2858036.2858308

[31] Bruno Fruchard, Eric Lecolinet, and Olivier Chapuis. 2017. MarkPad: Augmenting touchpads for command selection. In *Conference on Human Factors in Computing Systems - Proceedings*, Vol. 2017-May. 5630–5642. https://doi.org/10.1145/3025453.3025486

[32] H. Gellersen and F. Block. 2012. Novel Interactions on the Keyboard. *Computer* 45, 4, 36–40. https://doi.org/10.1109/MC.2012.112

[33] Renaud Gervais, Joan Sol Roo, and Martin Hachet. 2016. Tangible viewports: Getting out of flatland in desktop environments. In *TEI 2016 - Proceedings of the 10th Anniversary Conference on Tangible Embedded and Embodied Interaction*. 176–184. https://doi.org/10.1145/2839462.2839468

[34] Emmanouil Giannisakis, Gilles Bailly, Sylvain Malacria, and Fanny Chevalier. 2017. IconHK: Using Toolbar Button Icons to Communicate Keyboard Shortcuts. 12. https://doi.org/10.1145/3025453.3025595>

[35] Antonio Gomes, Tristan Trutna, and Roel Vertegaal. 2015. Display cover: A tablet keyboard with an embedded thin-film touchscreen display. In *17th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI 2015*. 531–535. https://doi.org/10.1145/2785830.2785843

[36] Keenan R Gray. 2018. *Facilitating Keyboard Use While Wearing a Head-Mounted Display*. Master's thesis. https://digitalcommons.wpi.edu/etd-theses/357

[37] Jens Grubert, Lukas Witzani, Eyal Ofek, Michel Pahud, Matthias Kranz, and Per Ola Kristensson. 2018. Effects of Hand Representations for Typing in Virtual Reality. *IEEE Virtual Reality (VR) 2018*. arXiv:1802.00613 http://arxiv.org/abs/1802.00613

[38] Jens Grubert, Lukas Witzani, Eyal Ofek, Michel Pahud, Matthias Kranz, and Per Ola Kristensson. 2018. Text Entry in Immersive Head-Mounted Display-based Virtual Reality using Standard Keyboards. In *IEEE Virtual Reality (VR) 2018*. arXiv:1802.00626 http://arxiv.org/abs/1802.00626

[39] Pelin Gul. [n.d.]. *Knowledge of the Microsoft Word Keyboard Shortcuts Expert vs. Novice Users*. Technical Report.

[40] Aldo Gunsing et al. 2021. KLFC: Keyboard Layout Files Creator. https://github.com/39aldo39/klfc

[41] SG Hart and LE Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human mental workload*. http://humanfactors.arc.nasa.gov/groups/TLX/downloads/NASA-TLXChapter.pdf

[42] Doris Hausen, Sebastian Boring, and Saul Greenberg. 2013. The unadorned desk: Exploiting the physical space around a display as an input canvas. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 8117 LNCS. 140–158. https://doi.org/10.1007/978-3-642-40483-2_10

[43] Anuruddha Hettiarachchi and Daniel Wigdor. 2016. Annexing Reality. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*. ACM Press, New York, New York, USA, 1957–1967. https://doi.org/10.1145/2858036.2858134

[44] Adrian H. Hoppe, Leonard Otto, Florian van de Camp, Rainer Stiefelhagen, and Gabriel Unmüßig. 2018. qVRty: Virtual Keyboard with a Haptic, Real-World Representation. Springer, Cham, 266–272. https://doi.org/10.1007/978-3-319-92279-9_36

[45] Brent E. Insko. 2019. OpenXR State of the Union. (2019). https://www.khronos.org/assets/uploads/developers/library/2019-gdc/OpenXR-Overview-GDC_Mar19.pdf

[46] Andrew F. Jarosz and Jennifer Wiley. 2014. What Are the Odds? A Practical Guide to Computing and Reporting Bayes Factors. *The Journal of Problem Solving* (2014). https://doi.org/10.7771/1932-6246.1167

[47] Shaun K. Kane, Daniel Avrahami, Jacob O. Wobbrock, Beverly Harrison, Adam D. Rea, Matthai Philipose, and Anthony LaMarca. 2009. Bonfire: A Nomadic System for Hybrid Laptop-Tabletop Interaction. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (Victoria, BC, Canada) *(UIST '09)*. Association for Computing Machinery, New York, NY, USA, 129–138. https://doi.org/10.1145/1622176.1622202

[48] Pascal Knierim, Valentin Schwind, Anna Maria Feit, Florian Nieuwenhuizen, and Niels Henze. 2018. Physical Keyboards in Virtual Reality: Analysis of Typing Performance and Effects of Avatar Hands. https://doi.org/10.1145/3173574.3173919

[49] Martin Krzywinski and Naomi Altman. 2014. Points of Significance: Visualizing samples with box plots. *Nature Methods* 11, 2 (jan 2014), 119–120. https://doi.org/10.1038/nmeth.2813

[50] Neeraj Kumar. 2015. Finger swipe gesture detection in C. https://github.com/neervfx/swipe

[51] Alton X Lam. 2018. ImageMSO Microsoft Office Icons (ImageMSO) Gallery & Extraction. https://archive.codeplex.com/?p=imagemso

[52] David M. Lane, H. Albert Napier, S. Camille Peres, and Aniko Sandor. 2005. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *International Journal of Human-Computer Interaction* 18, 2 (may 2005), 133–144. https://doi.org/10.1207/s15327590ijhc1802_1

[53] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. ACM, New York, NY, USA, Article 36, 17 pages. https://doi.org/10.1145/3173574.3173610

[54] Heejin Lee. 2018. Hain launcher. http://hainproject.github.io/hain/

[55] RV Lenth. 2018. Least-squares means: the R package lsmeans. *jstatsoft.org* (2018). https://www.jstatsoft.org/article/view/v069i01/v69i01.pdf

[56] Jia-Wei Lin, Ping-Hsuan Han, Jiun-Yu Lee, Yang-Sheng Chen, Ting-Wei Chang, Kuan-Wen Chen, and Yi-Ping Hung. 2017. Visualizing the Keyboard in Virtual Reality for Enhancing Immersive Experience. In *ACM SIGGRAPH 2017 Posters* (Los Angeles, California) *(SIGGRAPH '17)*. ACM, New York, NY, USA, Article 35, 2 pages. https://doi.org/10.1145/3102163.3102175

[57] David Lindlbauer, Jörg Müller, and Marc Alexa. 2017. Changing the Appearance of Real-World Objects by Modifying Their Surroundings. (2017). https://doi.org/10.1145/3025453.3025795

[58] Logitech, Inc. 2011. G19 Keyboard for Gaming - Logitech Support. https://support.logitech.com/en%5C_us/product/g19-keyboard-for-gaming/specs

[59] I. Scott MacKenzie and R. William Soukoreff. 2003. Phrase sets for evaluating text entry techniques. In *CHI '03 extended abstracts on Human factors in computing systems - CHI '03*. https://doi.org/10.1145/765891.765971

[60] Anindya Maiti, Murtuza Jadliwala, and Chase Weber. 2017. Preventing shoulder surfing using randomized augmented reality keyboards. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 630–635. https://doi.org/10.1109/PERCOMW.2017.7917636

[61] Sylvain Malacria, Gilles Bailly, Joel Harrison, Andy Cockburn, and Carl Gutwin. 2013. Promoting Hotkey Use Through Rehearsal with ExposeHK. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. ACM, New York, NY, USA, 573–582. https://doi.org/10.1145/2470654.2470735

[62] S.a Malacria, G.b Bailly, J.a Harrison, A.a Cockburn, and C.c Gutwin. 2013. Promoting hotkey use through rehearsal with ExposeHK. *Conference on Human Factors in Computing Systems - Proceedings*, 573–582. https://doi.org/10.1145/2470654.2470735

[63] Sylvain Malacria, Joey Scarr, Andy Cockburn, Carl Gutwin, and Tovi Grossman. 2013. Skillometers: Reflective widgets that motivate and help users to improve performance. In *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13*. ACM Press, New York, New York, USA, 321–330. https://doi.org/10.1145/2501988.2501996

[64] Mark McGill, Daniel Boland, Roderick Murray-Smith, and Stephen Brewster. 2015. A Dose of Reality: Overcoming Usability Challenges in VR Head-Mounted Displays. In *Proc. of CHI '15*. ACM Press, New York, New York, USA. https://doi.org/10.1145/2702123.2702382

[65] Mark Mcgill, Aidan Kehoe, Euan Freeman, and Stephen Brewster. 2020. Expanding the Bounds of Seated Virtual Workspaces. *ACM Trans. Comput.-Hum. Interact.* 27, 3, Article 13 (May 2020), 40 pages. https://doi.org/10.1145/3380959

[66] Craig S Miller, Svetlin Denkov, and Richard C Omanson. 2011. Categorization Costs for Hierarchical Keyboard Commands. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2765–2768. https://doi.org/10.1145/1978942.1979351

[67] Miscellaneous. 2018. AutoHotkey. https://www.autohotkey.com/

[68] Miscellaneous. 2018. Wox launcher for Windows. http://www.wox.one/

[69] Miscellaneous. 2019. Input System. https://valvesoftware.github.io/steamvr_unity_plugin/articles/SteamVR-Input.html

[70] Miscellaneous. 2021. Microsoft Keyboard Layout Creator. https://www.microsoft.com/en-us/download/details.aspx?id=102134

[71] Miscellaneous. 2021. Microsoft Mixed Reality Toolkit (MRTK). https://microsoft.github.io/MixedRealityToolkit-Unity/Documentation/WelcomeToMRTK.html

[72] Nemeio. 2019. The customisable, connected keyboard that uses electronic ink. https://www.nemeio.com/

[73] Jiamu Ni. 2017. *A Hotkey Interaction Technique that Promotes Hotkeys*. Master's thesis. https://aaltodoc.aalto.fi/bitstream/handle/123456789/28456/master_Ni_Jiamu_2017.pdf

[74] Benjamin Nuernberger, Eyal Ofek, Hrvoje Benko, and Andrew D. Wilson. 2016. *SnapToReality: Aligning Augmented Reality to the Real World*. Association for Computing Machinery, New York, NY, USA, 1233–1244. https://doi.org/10.1145/2858036.2858250

[75] 'Objective development Software'. 2018. LaunchBar. https://www.obdev.at/products/launchbar/index.html

[76] Oculus. 2020. Infinite Office. https://www.youtube.com/watch?v=5_bVkbG1ZCo

[77] Oculus. 2021. Infinite Office support for tracked K830 keyboard. https://www.oculus.com/blog/introducing-oculus-air-link-a-wireless-way-to-play-pc-vr-games-on-oculus-quest-2-plus-infinite-office-updates-support-for-120-hz-on-quest-2-and-more/

[78] Eyal Ofek, Jens Grubert, Michel Pahud, Mark Phillips, and Per Ola Kristensson. 2020. Towards a Practical Virtual Office for Mobile Knowledge Workers. arXiv:2009.02947 [cs.HC]

[79] Richard C Omanson, Craig S Miller, Elizabeth Young, and David Schwantes. 2010. Comparison of mouse and keyboard efficiency. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 54. Sage Publications Sage CA: Los Angeles, CA, 600–604.

[80] A. Otte, D. Schneider, T. Menzner, T. Gesslein, P. Gagel, and J. Grubert. 2019. Evaluating Text Entry in Virtual Reality using a Touch-sensitive Physical Keyboard. In *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. 387–392. https://doi.org/10.1109/ISMAR-Adjunct.2019.000-4

[81] Antti Oulasvirta and Andreas Karrenbauer. 2018. Combinatorial optimization for user interface design. In *Computational Interaction*. 97–119. https://doi.org/10.1093/oso/9780198799603.003.0005

[82] Henning Pohl, Christian Domin, and Michael Rohs. 2017. Beyond Just Text. *ACM Transactions on Computer-Human Interaction* 24, 1 (mar 2017), 1–42. https://doi.org/10.1145/3039685

[83] Henning Pohl and Michael Rohs. 2014. Around-device devices: My coffee mug is a volume dial. In *MobileHCI 2014 - Proceedings of the 16th ACM International Conference on Human-Computer Interaction with Mobile Devices and Services*. 81–90. https://doi.org/10.1145/

2628363.2628401

[84] Jon Porter. 2019. Report claims that Apple could begin production of iPhone-powered AR glasses this year. https://www.theverge.com/2019/3/8/18256256/apple-ar-glasses-2019-ming-chi-kuo-augmented-reality

[85] Ian Prest et al. 2021. KLE: Keyboard Layout Editor. http://www.keyboard-layout-editor.com/

[86] Razer. 2012. Razer DeathStalker Ultimate. https://support.razer.com/gaming-keyboards/razer-deathstalker-ultimate

[87] 'Running with Crayons'. 2018. Alfred - Productivity App for Mac OS X. https://www.alfredapp.com/

[88] Samsung.com. 2018. Samsung Odyssey Mixed Reality HMD. https://www.samsung.com/us/computing/hmd/windows-mixed-reality/xe800zaa-hc1us-xe800zaa-hc1us/

[89] Joey Scarr, Andy Cockburn, Carl Gutwin, and Philip Quinn. 2011. Dips and ceilings: Understanding and Supporting Transitions to Expertise in User Interfaces. *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*, 2741. https://doi.org/10.1145/1978942.1979348

[90] Daniel Schneider, Alexander Otte, Travis Gesslein, Philipp Gagel, Bastian Kuth, Mohamad Shahm Damlakhi, Oliver Dietz, Eyal Ofek, Michel Pahud, Per Ola Kristensson, Jorg Muller, and Jens Grubert. 2019. ReconViguRation: Reconfiguring Physical Keyboards in Virtual Reality. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. https://doi.org/10.1109/TVCG.2019.2932239

[91] Dominik Seger. 2013. *Not a Question of If, but When? Choosing the Right Trigger to Encourage Keyboard Shortcut Use.* Master's thesis. https://doi.org/10.1017/CBO9781107415324.004 arXiv:arXiv:1011.1669v3

[92] Kodai Sekimori, Yusuke Yamasaki, Yuki Takagi, Kazuma Murata, Buntarou Shizuki, and Shin Takahashi. 2018. Ex-Space: Expanded Space Key by Sliding Thumb on Home Position. In *International Conference on Human-Computer Interaction*. Springer, Cham, 68–78. https://doi.org/10.1007/978-3-319-91250-9_6

[93] Yilei Shi, Tomás Vega Gálvez, Haimo Zhang, and Suranga Nanayakkara. 2017. GestAKey: Get More Done with Just-a-Key on a Keyboard. In *Adjunct Publication of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST '17*. 73–75. https://doi.org/10.1145/3131785.3131786

[94] Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (aug 1983), 57–69. https://doi.org/10.1109/MC.1983.1654471

[95] Adalberto L Simeone, Eduardo Velloso, and Hans Gellersen. 2015. Substitutional Reality: Using the Physical Environment to Design Virtual Reality Experiences. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. https://doi.org/10.1145/2702123.2702389

[96] Tobias Sommer. 2018. Hotkey EVE. http://www.hotkey-eve.com/

[97] Marco Speicher, Anna Maria Feit, Pascal Ziegler, and Antonio Krüger. 2018. Selection-based Text Entry in Virtual Reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. https://doi.org/10.1145/3173574.3174221

[98] Art. Lebedev Studio. 2007. Optimus Maximus. https://www.artlebedev.com/optimus/maximus/

[99] Sussane Tak. 2007. *The Use of Keyboard Shortcuts Optimizing versus satisficing in the use of complex technology.* Master's thesis. http://www.efficiencysoftware.co.uk/uploads/nieuws/-1412774825.pdf

[100] Stuart Taylor, Cem Keskin, Otmar Hilliges, Shahram Izadi, and John Helmes. 2014. Type-hover-swipe in 96 bytes: a motion sensing mechanical keyboard. *CHI '14: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1695–1704. https://doi.org/10.1145/2556288.2557030

[101] Bruce Tognazzini. 1989. Keyboard vs. The Mouse, pt1. https://www.asktog.com/TOI/toi06KeyboardVMouse1.html

[102] Jo Vermeulen, Kris Luyten, Elise van den Hoven, and Karin Coninx. 2013. Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. ACM, New York, NY, USA, 1931–1940. https://doi.org/10.1145/2470654.2466255

[103] VIVE Blog. 2018. Introducing the Logitech BRIDGE SDK. https://blog.vive.com/us/2017/11/02/introducing-the-logitech-bridge-sdk/

[104] J. A. Wagner Filho, C.M.D.S. Freitas, and L. Nedel. 2018. VirtualDesk: A Comfortable and Efficient Immersive Information Visualization Approach. *Computer Graphics Forum* 37, 3, 415–426. https://doi.org/10.1111/cgf.13430

[105] James Walker, Bochao Li, Keith Vertanen, and Scott Kuhl. 2017. Efficient Typing on a Visually Occluded Physical Keyboard. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*. 5457–5461. https://doi.org/10.1145/3025453.3025783

[106] James A. Walsh, Stewart von Itzstein, and Bruce H. Thomas. 2014. Ephemeral Interaction Using Everyday Objects. In *Proceedings of the Fifteenth Australasian User Interface Conference - Volume 150* (Auckland, New Zealand) *(AUIC '14)*. Australian Computer Society, Inc., AUS, 29–37.

[107] Windows Experience Blog. 2018. Windows 10 Tip: Get started with the emoji keyboard shortcut. https://blogs.windows.com/windowsexperience/2018/02/05/windows-10-tip-get-started-emoji-keyboard-shortcut/

[108] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. 2011. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*. ACM Press, New York, New York, USA, 143. https://doi.org/10.1145/1978942.1978963

[109] Chien-Min Wu, Chih-Wen Hsu, Tzu-Kuei Lee, and Shana Smith. 2017. A virtual reality keyboard with realistic haptic feedback in a fully immersive virtual environment. *Virtual Reality* 21, 1 (mar 2017), 19–29. https://doi.org/10.1007/s10055-016-0296-6

[110] Robert Xiao, Julia Schwarz, Nick Throm, Andrew D Wilson, and Hrvoje Benko. 2018. MRTouch: Adding Touch Input to Head-Mounted Mixed Reality. *IEEE transactions on visualization and computer graphics* (2018). https://doi.org/10.1109/TVCG.2018.2794222

[111] Naoki Yanagihara, Buntarou Shizuki, and Shin Takahashi. 2019. A Comparative Study of Planar Surface and Spherical Surface for 3D Pointing Using Direct Touch. In *25th ACM Symposium on Virtual Reality Software and Technology* (Parramatta, NSW, Australia) *(VRST '19)*. Association for Computing Machinery, New York, NY, USA, Article 42, 2 pages. https://doi.org/10.1145/3359996.3364814

[112] Zhizhuo Yang, Dongdong Weng, Zhengliang Zhang, Yufeng Li, and Yue Liu. 2016. Perceptual Issues of a Passive Haptics Feedback Based MR System. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*. IEEE, 310–317. https://doi.org/10.1109/ISMAR-Adjunct.2016.0103

[113] Haimo Zhang and Yang Li. 2014. GestKeyboard: Enabling Gesture-Based Interaction on Ordinary Physical Keyboard. *Proceedings of CHI 2014*, 1675–1684. https://doi.org/10.1145/2556288.2557362

[114] Jingjie Zheng. 2017. *Enabling Expressive Keyboard Interaction with Finger, Hand, and Hand Posture Identification.* Master's thesis. University of Waterloo. https://doi.org/10.1145/2858036.2858355

[115] Jingjie Zheng and Daniel Vogel. 2016. Finger-Aware Shortcuts. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*. 4274–4285. https://doi.org/10.1145/2858036.2858355

[116] Ryder Ziola, Melanie Kellar, and Kori Inkpen. 2007. DeskJockey: Exploiting Passive Surfaces to Display Peripheral Information. In *Human-Computer Interaction - INTERACT 2007*, Vol. 4662. 447–460. https://doi.org/10.1007/978-3-540-74796-3_43