Evangelista Belo, João Marcelo; Lystbæk, Mathias N.; Feit, Anna Maria; Pfeuffer, Ken; Kán, Peter; Oulasvirta, Antti; Grønbæk, Kaj

AUIT - the Adaptive User Interfaces Toolkit for Designing XR Applications

Published in:
UIST 2022 - Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology

DOI:
10.1145/3526113.3545651

Published: 29/10/2022

Document Version
Publisher's PDF, also known as Version of record

AUIT – the Adaptive User Interfaces Toolkit for Designing XR Applications

João Belo
joaobelo@cs.au.dk
Aarhus University
Denmark

Mathias N. Lystbæk
Aarhus University
Denmark

Anna Maria Feit
Saarland University, Saarland Informatics Campus
Germany

Ken Pfeuffer
Aarhus University
Denmark

Peter Kán
Vienna University of Technology
Austria

Antti Oulasvirta
Aalto University
Finland

Kaj Grønbæk
Aarhus University
Denmark

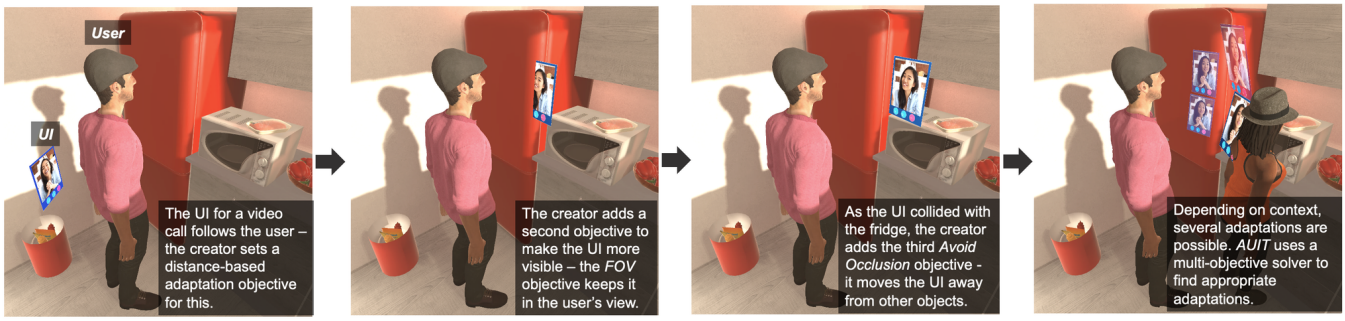


Figure 1: AUIT supports creators defining adaptation policies for UI elements that combine multiple objectives for XR interfaces. In this example, a video call UI is gradually extended with adaptation objectives to render it visible and within reach. Complexity rises with more potentially *competing* objectives and context changes. AUIT simplifies the design of adaptations by finding the best compromise via a multi-objective solver.

ABSTRACT

Adaptive user interfaces can improve experiences in Extended Reality (XR) applications by adapting interface elements according to the user's context. Although extensive work explores different adaptation policies, XR creators often struggle with their implementation, which involves laborious manual scripting. The few available tools are underdeveloped for realistic XR settings where it is often necessary to consider conflicting aspects that affect an adaptation. We fill this gap by presenting AUIT, a toolkit that facilitates the design of optimization-based adaptation policies. AUIT allows creators to flexibly combine policies that address common objectives in XR applications, such as element reachability, visibility, and consistency. Instead of using rules or scripts, specifying adaptation policies via adaptation objectives simplifies the design process and enables creative exploration of adaptations. After creators decide which adaptation objectives to use, a multi-objective solver finds appropriate adaptations in real-time. A study showed that AUIT allowed creators of XR applications to quickly and easily create high-quality adaptations.

CCS CONCEPTS

• **Human-centered computing** → **Systems and tools for interaction design**; *Gestural input*; **Mixed / augmented reality**; *Virtual reality*; **User interface toolkits**.

KEYWORDS

extended reality, multi-objective optimization, adaptive user interfaces, toolkit, context-awareness

ACM Reference Format:
João Belo, Mathias N. Lystbæk, Anna Maria Feit, Ken Pfeuffer, Peter Kán, Antti Oulasvirta, and Kaj Grønbæk. 2022. AUIT – the Adaptive User Interfaces Toolkit for Designing XR Applications. In *The 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22), October 29-November 2, 2022, Bend, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3526113.3545651

UIST '22, October 29-November 2, 2022, Bend, OR, USA

ACM ISBN 978-1-4503-9320-1/22/10.
https://doi.org/10.1145/3526113.3545651

1 INTRODUCTION

Extended Reality (XR) is a medium that has gotten more widespread over the past years and will likely continue growing in the years to come [10]. Hardware improvements push the boundaries of what these applications can achieve, and sectors such as entertainment and manufacturing contribute towards this computing platform increasing popularity. However, easy-to-use XR applications are still challenging to develop. In contrast to traditional desktop or mobile applications, they are not confined to a 2D screen, but merge with the user's real-world environment to different extents. A key challenge of XR applications is how well they adapt to changes in the user's situation and surroundings [37].

The design of an adaptive UI for XR applications involves a high degree of complexity. As the user moves in the environment, considering context changes like his position or surrounding objects is crucial for creating an adaptation policy that provides a usable UI. Figure 1 illustrates one example scenario where a user has a floating video call interface close to him. The user might be unable to reach the virtual buttons, and positions outside his field of view or colliding with physical objects are inappropriate. Thus, the environment's geometry and the user's position constantly affect the *visibility* and *reachability* of the UI element - two fundamental usability factors of XR applications.

To address both requires considering multiple adaptation objectives [17]. However, these are typically not independent and might compete with each other. For example, moving the video call to prevent occlusion might position it outside the reach of a user. Such interactions grow as the number of UI elements and the complexity of the environment increase. They are hard for developers to foresee and resolve, increasing the difficulty of creating adaptive XR interfaces.

Over the last years, HCI researchers have proposed various methods to adapt interface elements in XR applications. They were concerned with the visibility and integration of virtual elements into the physical environment [9, 23, 37, 52] and their reachability or ergonomics [16, 30]. When considering criteria to adapt, these typically address independent aspects of the interface, such as position and content [37]. However, these methods tend to be custom tailored to specific applications and are difficult for creators to implement in practice. Existing tools for developing XR applications [45] only offer naive adaptation policies that are ineffective when multiple usability aspects come together.

To close this gap, we propose AUIT, the **A**daptive **U**ser **I**nterfaces **T**oolkit for supporting the design of XR applications. AUIT simplifies the adaptation of virtual elements to users' contexts and enables the combination of multiple adaptation objectives. It also offers a general framework that unifies prior research to make it available to practitioners. We achieve this goal by identifying five design concepts that adaptive user interfaces must implement:

Adaptation objectives Describe adaptation behaviors to address, such as visibility and reachability of UI elements.

Solvers Algorithms to compute adaptation candidates for UIs, resolving conflicts between objectives.

Context widgets Process raw sensor data into higher abstraction levels to inform adaptations.

Adaptation triggers The logic for when to invoke solvers and when to apply the adaptation proposals to the UI.

Property transitions How properties of virtual content transition to a new state when adaptations are triggered.

AUIT implements seven *adaptation objectives* that creators can flexibly assign to UI elements to address two fundamental usability issues of XR interfaces: visibility and reachability. AUIT automates conflict resolution by continuously optimizing the interface and determining the best trade-off between the chosen adaptation objectives using a multi-objective *solver*. Creators can choose between different *adaptation triggers* for initiating the adaptation, either at fixed time intervals or when the solver finds substantial UI improvements. They can also select *property transitions* to decide how the UI transitions to its new state. AUIT is implemented as a Unity extension that creators can easily import to develop adaptive UIs without drastic changes in their current workflow.

We evaluate AUIT's usefulness through a user study with eight experts who actively create XR applications as part of their jobs. We found that the design concepts in AUIT were easy to understand for participants, allowing for a clear separation of concerns in adaptive UIs. The process was fast, and participants designed adaptive user interfaces for two different scenarios in less than 25 minutes. They appreciated how easy it was to combine adaptation objectives and the quality of the results, while feeling efficient considering the time spent and the adaptations obtained. Participants also discussed the importance of adaptive UIs and pointed out that their current practice was limited by manual scripting, highlighting the need for tools to facilitate their development.

To summarize, this paper proposes AUIT, a toolkit based on a conceptual framework to support the creation of user interfaces that adapt to the user's context. It allows for 1) combining different adaptation objectives, 2) resolving conflicts between objectives using multi-objective optimization, and 3) customizing how and when XR content adapts. We demonstrate the utility of the toolkit through a study where experts successfully create high-quality adaptations for two applications using AUIT. We make the toolkit available through a Unity package that can be extended and customized by creators to fit their needs. AUIT makes existing research on adaptation methods for 3D interfaces available to creators, and offers a unifying framework for future work. Source code is available at https://github.com/joaobelo92/auit.

2 RELATED WORK

Over the last decades, researchers have proposed different methods to adapt interfaces to improve usability. We start with a brief overview of adaptation and optimization techniques for 2D user interfaces. Then, we move on to research focusing on XR, starting with view management techniques, followed by adaptation techniques focusing on other usability goals. Finally, we give an overview of related frameworks and toolkits.

2.1 Adaptation and Optimization of 2D Interfaces

The increasing availability of mobile devices has motivated significant work on adaptive UIs. Researchers have proposed model-based

approaches allowing developers to adapt applications across devices based on rules and logic (e.g., MARIA/TERESA [47, 56]) and methods that dynamically generate interfaces for multiple devices [50]. Gajos and Weld introduced SUPPLE, an approach that uses optimization to design UIs [21]. Similarly to SUPPLE, we use cost functions in our optimization procedure to represent objectives that guide adaptations in XR.

To support designers of 2D applications, researchers have explored genetic algorithms [60], other combinatorial optimization approaches [54], and data-driven optimization [22] based on user preferences to compute an optimal UI. The UI can also use optimization in real-time to dynamically adapt to users' preferences, context, or a device's capabilities (e.g. [8, 11, 18, 25, 55]). Such adaptations of user interfaces are relevant on mobile devices [5, 51] that typically have small screen sizes [20]. Recently, Todi et al. [65] presented a method for adaptive user interfaces based on reinforcement learning. There is extensive research on adaptive 2D UIs, and we refer to Miraz et al. [46] for a broader discussion of related work in this area.

2.2 View Management Techniques

View management techniques address how to maintain virtual objects in the user's view plane. These techniques focus on visibility aspects, such as avoiding occlusion and maintaining spatial relationships of virtual objects. Pioneering work focused on algorithms that use the upright rectangular extents of content in the view plane to avoid occlusion, adapting object properties such as their position, size, and transparency [3]. Grasset et al. focused on optimizing layouts of elements during run-time in the 2D view plane [26], while Tatzgern et al. explored this issue in 3D space [63]. Spatio-temporal coherence is another relevant factor to consider in XR experiences. Using spatial information from previous frames can help reduce visual discontinuities. Experiments suggest that users prefer limited update rates over continuous update rates for adaptations [40].

Other work investigated adaptive UIs to manage information density in AR and avoid information overload, which might affect task performance depending on the user's cognitive load. Therefore, researchers have developed adaptive level-of-detail (LOD) methods for AR interfaces. Tatzgern et al. [64] proposed an adaptive information density display for AR using hierarchical clustering. Their approach automatically groups UI elements to reduce information overload and provides the user the control to unfold the level of detail. Several works use special sensors, such as eye-tracking technology, to adapt how and which content to present to the user [37, 38, 57].

View management techniques are closely related to UI adaptations. AUIT aims to make this line of work available to practitioners through a tool to adapt UIs that they can extend to support other sources of context (e.g., eye-gaze) and objectives (e.g., less-cluttered UI).

2.3 Adaptive User Interfaces in XR

UIs in XR pose additional challenges compared to traditional UIs because of the higher-dimensional design space, context changes, and broader range of interaction metaphors. Adaptive UIs are particularly important in XR scenarios using wearable [34], ubiquitous [29, 61], and mobile [31] computing platforms. Oliveira and Araujo [53] developed a context-aware AR system that adapts its interface based on changing contexts. Their system uses adaptation rules which select an appropriate UI pattern according to the current context. To improve the usability of XR applications, creators must consider factors such as real-world geometry [23, 52], cognitive load [37], or ergonomics [16]. Gal et al. [23] presented a method to automatically generate object layouts in AR applications, where the virtual elements in the AR application adapt to real-world geometry.

Ens et al. proposed a body-centric layout management technique that keeps layouts consistent across multiple environments while adapting to local geometric and visual features [15]. The work from Xiao et al. [69] explores various interaction techniques that use spatial awareness and optimization to adapt to different work surfaces. Later on, Lindlbauer et al. proposed an optimization-based approach to automatically control when and where mixed reality (MR) applications are shown and how much information they display, depending on the user's cognitive load [37]. Lu and Xu [39] studied different levels of automation and control of adaptive UI in AR. Their results suggest that users prefer and perform better when adaptations are semi-automated.

All these works show how different adaptation factors can improve usability in XR applications. AUIT provides a novel platform for creators to experiment with several adaptation goals and can be extended to support many more.

2.4 Frameworks and Toolkits

As XR technology is becoming widely available, researchers called for better support for developers across various stages of the design process of creating XR experiences [2]. As such, there has been a surge in research for tools that can ease the design and development of augmented [36, 48, 62] and virtual reality applications [27, 49]. We extend existing work with a toolkit to create adaptive UIs.

There is limited work on frameworks to facilitate the creation of adaptive UIs. Bonanni et al. [4] presented a framework for adaptive UI design focused on an AR kitchen scenario to support cooking, a scenario we build upon in our user study. Krings et al. implemented context-aware UI adaptations with a rule-based framework in which any change in context can trigger adaptation actions [33].

There are also other frameworks with some support for creating UI adaptations. For example, MRTK [45] has solvers [43] that use algorithms to calculate the position and orientation of UI elements. Existing solvers in MRTK focus on fundamental usability issues, such as visibility and reachability, as we do in the initial iteration of our toolkit. Although MRTK allows creators to chain multiple solvers with different adaptation objectives, it runs these sequentially without support for multi-objective optimization. Moreover, each solver in MRTK is tied to specific transitions (e.g., smooth movement over time, triggered every frame), limiting the design space of adaptations for XR.

Unity Mars [67] is another authoring tool that provides proxies to represent real-world objects, allowing creators to design UI adaptations based on rules relative to these proxies.

These frameworks focus on rule-based adaptations or algorithms to address specific adaptation objectives and lack the flexibility to combine multiple adaptation goals. We propose a framework that can integrate existing research and is easy to extend and generalize various scenarios encountered in the XR landscape. We use this framework as the foundation for AUIT, giving flexibility to creators by allowing them to combine different adaptation objectives, find adaptations using multi-objective optimization, and customize when and how UI elements transition from one state to another. AUIT separates adaptation concerns [13] into components, providing a modular approach where it is straightforward to customize different aspects of an adaptation.

3 AUIT: DESIGN CONCEPTS

To facilitate the design of adaptive UIs, we propose a clear separation of concerns [13] of the different design concepts present in an adaptation. Throughout this paper, we refer to the **creator** as the individual responsible for the application's development or design and the **user** as the end-user that will use the application. Consider the video call scenario presented in Figure 1. A creator develops an application that consists of a single UI element with a live video call and some controls to interact with it. The creator wants the video to be visible to the user and follow him as he moves around without interfering with his tasks and the environment. Such a scenario can quickly become complex, with various considerations about what kind of adaptive behavior is required, what contextual information it depends on, the conditions that cause an adaptation, and how to execute the UI adaptation. Such questions are not specific to this example, and are relevant to consider for many types of UI adaptations in XR. Therefore, we formulate these concerns as design goals for UI adaptations, followed by a framework to address them.

3.1 Design Goals

D1: Support a range of adaptation behaviors. XR applications are not limited to the dimensions of a screen, in contrast to the GUIs present in traditional applications. In this setting, the design space tends to be broad and challenging to predict at design time. The design space also changes at runtime due to context changes such as the user's position or moving real-world objects. Consider the scenario in Figure 1 - the creator designs an adaptive UI that is 1) in reach, 2) in the user's field of view, and 3) not occluded by other objects. However, different scenarios have different requirements, and no specific combination of adaptation behaviors addresses the needs of the wide variety of applications possible in XR. Lindlbauer et al. [37] explored multiple adaptation objectives, but their approach focus on specific scenarios and fixed adaptation objectives, limiting generalization to other settings. Flexibility to combine different adaptation objectives across various UI elements allows creators to develop a wider variety of designs.

D2: Allow combining multiple adaptation objectives in one adaptation. Multiple adaptation objectives can conflict with each other. For example, in the scenario from Figure 1, the adaptation objective to position objects in a specific zone of the user's field of view (FoV) can conflict with the objective to avoid collisions. Although related work has explored methods to find suitable solutions when considering multiple adaptation objectives through multi-objective optimization [23, 37], the support for combining adaptation objectives is still limited in existing tools. Therefore, the framework must be capable of finding a compromise between multiple objectives at runtime without constraining which objectives are possible to select.

D3: Support for context collection and interpretation. The lack of standard methods to acquire and handle context is one of the barriers identified by Dey et al. [12] for using context in applications. In the scenario from Figure 1, context plays a crucial role in providing appropriate UI adaptations - it is necessary to know the user's position, where he is looking, and the environment geometry. In the past, applications would retrieve the user's context with custom implementations that processed sensor data into applications. Nowadays, this issue is not as prominent for XR applications. Game engines and software development kits already support some contextual information at a high abstraction level. Nonetheless, methods to interpret context at higher levels of abstraction that are generalizable across different applications are still a requirement to facilitate the creation of adaptive UIs in XR.

D4: Methods to customize when and why an adaptation occurs. Depending on the application, creators might require different strategies for triggering UI adaptations. For example, in the video call scenario encountered in Figure 1, a naive approach that adapts the UI at a constant update rate might be sufficient, but the creator could be interested in a different strategy such as adapting the UI only when the quality of the layout goes below a certain threshold. Lindlbauer et al. [37] propose temporal smoothing to improve transitions through adaptations, while Krings et al. [33] decide when to adapt the UI based on rules. The framework must allow creators to customize why and when UI adaptations occur to increase flexibility.

D5: Support for a variety of property transitions. When considering the position of a UI element in an XR application, there are many possible ways it can adapt from one state to another. For example, in Figure 1, a creator can choose to update the position of the video call by moving it over time in 3D or instantly. These are just a few of the many possible transitions a creator could use for adapting the position of an object, one of the properties to adapt in XR. Consider now other properties of UI elements, like size, rotation, or modality. In such a vast design space, a framework to facilitate the creation of adaptive user interfaces must allow creators to choose from multiple property transitions.

3.2 Design Concepts for Adaptation Policies

We propose five design concepts for the development of adaptation policies to address the design goals we just presented. We provide an overview in Figure 2. Creating an adaptation policy that considers multiple adaptation objectives for an XR application should incorporate these to some extent:

Adaptation objectives (D1) are goals that guide the UI adaptation. For greater flexibility, an objective should only have one goal, allowing creators to combine objectives with different goals in one UI adaptation. For example, a creator might want a button to be reachable to the user while avoiding collisions with the real-world environment. By abstracting these goals into two separate

objectives, creators can use them modularly for other adaptations throughout the application. Although an adaptation objective must have a single goal, it is worth noting that it can refer to a set of UI elements. For example, an objective to avoid clutter can have multiple UI elements as a target, but it is still a single goal.

Solvers (D2) are approaches that try to find the optimal solution to a stated optimization problem [54]. In our framework, solvers generate adaptation proposals to optimize the UI according to the adaptation objectives selected by creators.

Context widgets (D3) encapsulate how context is retrieved and make that data accessible to applications. Dey et al. [12] proposed such a component, and we refer to their work for a more in-depth overview. In short, context widgets process raw data and make it available at higher levels of abstraction, allowing creators to reuse and customize the usage of context data throughout the application. For XR applications, development tools such as MRTK [45] have some context widgets available. An example is the spatial awareness system in MRTK [44], a feature to provide real-world environmental awareness through a collection of meshes representing the environment geometry, which demonstrates how raw sensor data from the device is converted into a higher level of abstraction (in a mesh format), facilitating its integration in XR applications.

Adaptation Triggers (D4) are responsible for the logic to invoke solvers and if the solver proposals are applied. For example, creators can save computational resources by invoking the solver only when the layout quality goes below a certain threshold. Then, adaptations might be applied if the improvements from a new proposal are sufficient to justify the adaptation. The framework should allow creators to customize adaptation triggers and use or implement different strategies.

Property Transitions (D5) address how virtual content adapts to its new state when adaptations occur. Once an adaptation trigger executes an adaptation, property transitions define how the relevant properties of the UI element adapt from the previous to the new state. For example, there are different ways a UI element can move from position x to position y, such as moving over time from one position to another or fading out from the previous to the new position.

4 AUIT: TOOLKIT IMPLEMENTATION

We implement the framework introduced in the prior section through AUIT, a toolkit to facilitate the creation of adaptive user interfaces for XR applications. To optimize XR interfaces considering a combination of adaptation objectives, we formulate a cost minimization problem and solve it using multi-objective optimization. Adaptation objectives are formulated mathematically through a cost function representing how much the current layout fulfills that objective.

From an optimization perspective, we are dealing with a multi-objective optimization problem to optimize multiple objective functions simultaneously. In this case, objectives can contradict each other such that improving the solution towards one will worsen any of the others. Non-trivial problems have a set of optimal solutions that form the Pareto optimal frontier [41] instead of a single global optimal solution. To simplify picking a desirable solution in our toolkit, we opt for a weighted sum method [41], where creators

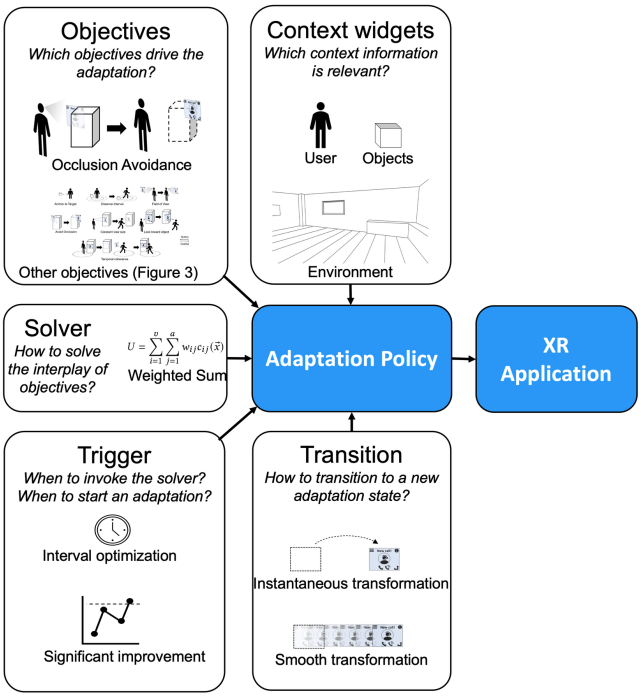


Figure 2: Overview of the design concepts for adaptive UIs. MAUI proposes 5 concepts to design adaptation policies for XR applications.

articulate their preferences about the relative importance of different objectives using weights. We describe how we implement each design concept we proposed in Section 3 as a component of our toolkit:

4.1 Adaptation Objectives

Adaptation objectives are the criteria the UI adapts to and represent atomic adaptation behaviors that accomplish usability goals, such as visibility and reachability of the UI.

We define each adaptation objective through a cost function. To facilitate the customization of weights by creators, each adaptation objective we implement has a normalized cost function that outputs a cost from 0 to 1, reaching the highest value when the current layout infringes the adaptation objective beyond a customizable threshold. For example, consider an adaptation objective to keep virtual content from colliding with objects in the physical world. In this context, such an objective would return a value of 0 when applied to a hologram occupying a position that results in no collisions. This value would increase when the hologram starts colliding with environment geometry, reaching the value of 1 when the whole area of its virtual content is colliding. We implement heuristics for each adaptation objective so the solver can find improvements more efficiently. In addition, the solver can still search for new solutions following a random approach to avoid getting stuck in local minima.

We include seven adaptation objectives in AUIT that we illustrate in Figure 3, that identifies the high-level usability goals each

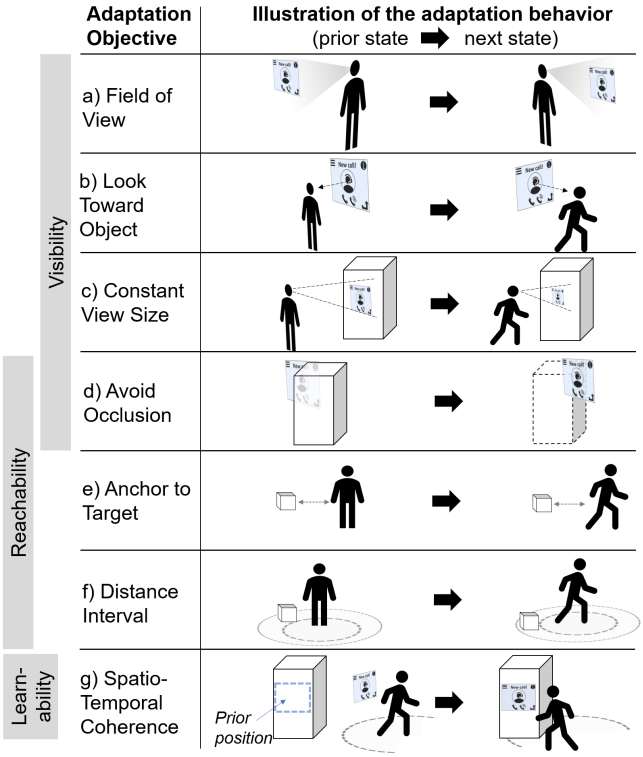


Figure 3: Adaptation objectives that are currently supported in AUIT. Creators can customize and combine them to design adaptation policies.

adaptation objective contributes to, from visibility, reachability, and learnability. Here, we briefly describe the adaptation objectives AUIT supports and refer the reader to the appendix for a more detailed description of cost functions and optimization heuristics.

4.1.1 Field of View Objective (Figure 3a). Ensures the UI element is within a specific region of the user's field of view. Creators can select a pre-defined peripheral view interval or create a custom one by defining its inner and outer boundaries.

Optimization heuristic: attempt to move the UI element towards the FoV interval selected by the creator.

4.1.2 Look Towards Objective (Figure 3b). Rotates the UI element towards a selected context source. It defaults to the user's position. This objective can contribute to visibility, as content such as text and images will become easier to see when rotated towards the user.

Optimization heuristic: rotate the UI towards the optimal rotation.

4.1.3 Constant View Size Objective (Figure 3c). Scales the UI element depending on its distance from a target. This objective aims to maintain a constant view size to a context source, typically the user. We determine the optimal size of the UI using a configurable linear function dependent on the distance from the UI to the context source. This is relevant to improve visibility of the UI without updating its position.

Optimization heuristic: scale the UI towards the optimal scale according to its distance from the context source.

4.1.4 Avoid Occlusion Objective (Figure 3d). Avoids positions where the environment geometry or other virtual elements in the scene would occlude the object (typically to the user). Avoiding occlusions also prevents collisions with other virtual or physical objects. To check for occlusion, we dynamically add a configurable set of points in a grid composition to the UI element. Then, we draw rays [59] from the context source to each point and increase the cost for each ray hitting other content.

Optimization heuristic: move the UI in the direction of the surface normal hit by one of the obstructed rays.

4.1.5 Anchor to Target Objective (Figure 3e). Aims to position a UI element at an offset from a selected context source. It selects the rotation and position of the user's head by default and the creator must provide the offset. Relevant when a UI element should follow the user or a virtual object.

Optimization heuristic: move the UI in the direction of the anchor point by a random distance.

4.1.6 Distance Interval Objective (Figure 3f). Keeps UI elements positioned within a customizable distance interval in the shape of a vertical hollow cylinder from a selected context source - typically the user's position. The creator can set the inner and outer boundary of the cylinder area. This objective is relevant in cases where UI elements must be reachable to support hand input or at a distance that allows the user to see their content.

Optimization heuristic: move the UI towards the hollow cylinder by a random distance.

4.1.7 Spatio-Temporal Coherence Objective (Figure 3g). Prioritizes adaptations where the UI element adapts to positions where it has been before, avoiding updates unless there are substantial improvements. Relevant to improve usability by leveraging the user's spatial memory [40].

Optimization heuristic: pick the closest previously visited position or other visited position at random.

4.2 Solvers

Solvers are the algorithms responsible for computing adaptation proposals. As mentioned earlier, we formulate a cost minimization problem and allow creators to articulate their preferences in terms of the relative importance of each adaptation objective using weights. An XR application typically contains v virtual elements. Each virtual object can have multiple adaptation objectives a - each having a correspondent weight w and cost function c - we can articulate the problem with a weighted sum:

$$U = \sum_{i=1}^{v} \sum_{j=1}^{a} w_{ij} c_{ij}(\vec{x}) \quad (1)$$

where $\vec{x}$ is the decision vector that consists of UI configuration parameters for all the UI elements to optimize and the minimum of U is Pareto optimal [41]. One of the goals of our framework is to generalize to a wide range of objectives, so we are particularly interested in methods capable of solving non-linear optimization problems. For that reason, the solver we implement uses simulated

annealing [32, 68], which gradually converges to a near-optimal solution [1]. To find appropriate solutions more efficiently, our solver uses heuristics implemented for each objective and some randomness to avoid local optima. The solver uses early stopping to stop searching for proposals when it finds a suitable candidate and Unity coroutines [66] to distribute the computational load across multiple frames.

4.3 Context Widgets

Context widgets are the components responsible for processing raw sensor data into higher levels of abstraction. An advantage of building our toolkit for Unity is that several context widgets are available out of the box. It is trivial to retrieve fundamental context data for XR applications such as the position and gaze of the user from the Unity Camera component, which follows the user's head movement and rotation when using an HMD. Other relevant context information, such as the user's environment geometry or hand tracking, can be available depending on the development platform. For example, when developing for the Hololens, its SDK provides hand tracking and the environment geometry in Unity [44]. Because the adaptation objectives that are part of our first iteration of the toolkit consider fundamental usability goals, the context widgets already available in Unity are sufficient. For more flexibility, the creator can change the context source of adaptation objectives in the Unity inspector. For example, an adaptation objective that uses a position in 3D can be customized to use the user's pose or another virtual object in the scene.

4.4 Adaptation Triggers

Adaptation triggers are the component responsible for the logic to invoke the solver and apply the resulting adaptation proposal. AUIT supports two adaptation triggers to handle 1) when and how frequently to invoke the solver and 2) when to apply the adaptation proposed by the solver:

4.4.1 Interval optimization trigger. A basic adaptation strategy is to use a solver to generate UI proposals and apply them at a fixed rate, which creators can customize. This strategy involves a trade-off between update rate *vs.* usability. If the update rate is too high, it can result in too many adaptations that are a nuisance to the user. If the update rate is too low, the UI might violate adaptation objectives for too long until it adapts. Moreover, higher update rates result in a higher computational load as the solver runs more often.

4.4.2 Significant improvement trigger. This adaptation trigger only adapts the UI if it will result in an improvement. It invokes the solver once the quality of the UI (determined by the correspondent cost functions) is below a configurable threshold. Adaptations are applied if they improve on the previous interface by a ratio customizable by creators. This approach saves computational power because the solver only executes when the quality of the UI declines.

4.5 Property Transitions

Property transitions adapt the UI from one state to another. Once an adaptation trigger starts an adaptation for a UI element, its property transitions are applied. Depending on the property to adapt, AUIT invokes the correspondent property transition - different properties

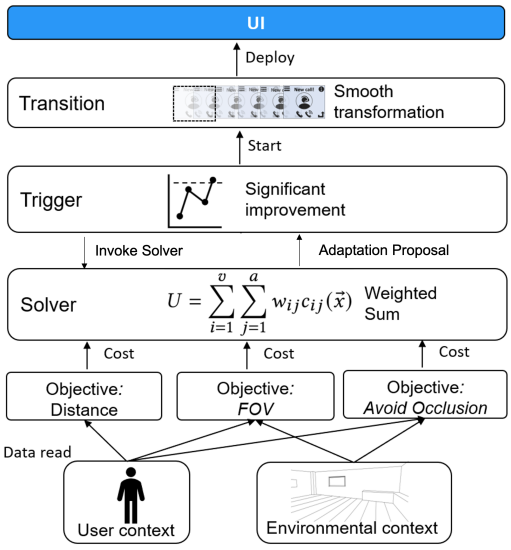


Figure 4: The processing flow of AUIT for the example adaptation in Figure 1. The cost from each objective is calculated using context. When invoked, the solver computes a new adaptation proposal considering all adaptation objectives – if this significantly improves the adaptation state, the UI uses a smooth movement transition to accomplish the UI adaptation. A solver can find proposals for multiple UI elements in the same optimization loop.

such as position and rotation require their respective property transitions. AUIT supports the following:

4.5.1 Instantaneous Movement. UI element moves instantaneous from the current position to the new one.

4.5.2 Smooth Movement. movement animation from the current to the new position over time using linear interpolation.

4.5.3 Smooth Rotation. rotation animation from the current to the new rotation over time using linear interpolation.

4.5.4 Smooth Scaling. scaling animation from the current to the new scale over time using linear interpolation.

4.6 Architecture

Now we describe the software architecture to put all the components in AUIT together (see Figure 5).

Adaptation objectives access their context widgets directly to compute cost functions. The toolkit allows creators to change the context widget that is used by an adaptation objective through the Unity inspector, as long as it is compatible (e.g., anchor to target objective can use the user's position or the position of another virtual object as its context source). Adaptation objectives are then associated with UI elements. It is possible to associate the same adaptation objective to different UI elements, and each instance supports different configurations. Creators must assign property

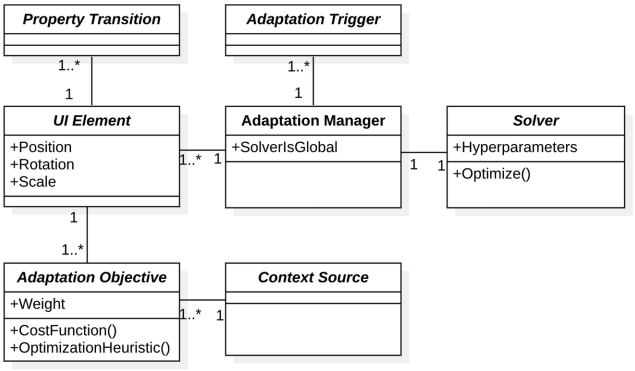
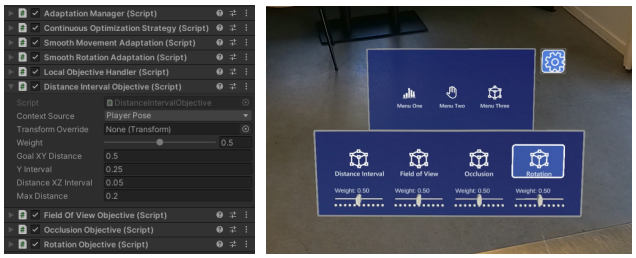


Figure 5: AUIT software architecture



(a) In-Editor Work **(b) In-scene Menus**

Figure 6: Users primarily use AUIT through the editor, but it is possible to tweak weights in an immersive setting for online configuration of adaptations.

transitions for each property the UI elements adapt to (e.g., adaptations that involve the position and rotation of the UI will require corresponding property transitions).

To manage adaptation triggers and solvers, we created an auxiliary class named Adaptation Manager. Adaptation managers gather all the objectives in the UI elements to optimize, invoke the solver using the logic in the adaptation trigger to compute adaptation proposals, and apply adaptations using the property transitions. Note that adaptation managers can optimize multiple UI elements in the same optimization loop by enabling a flag indicating the solver is global and indicating which UI elements it optimizes.

The adaptation objectives, triggers, and transitions derive from a corresponding abstract class. These abstract classes serve as a starting point for creators that want to extend the toolkit with new implementations of the components present in AUIT. Implementing additional components for AUIT, such as new adaptation objectives or property transitions, can be done by following three steps:

(1) Create a new script component in Unity
(2) Inherit from the abstract class that implements the component of interest
(3) Implement the correspondent class abstract methods (e.g., for Adaptation Objectives, implement the *Cost Function* and *Heuristic* methods)

4.7 Technical Performance

Although not the focus of this work, it is important to assess how well it performs on common XR devices. Hence, we benchmark how the toolkit runs on a standalone device tailored for MR experiences. We test the Microsoft HoloLens 2 [42], a popular MR device nowadays. Note that the HoloLens 2 uses a mobile Qualcomm Snapdragon 850 with limited performance compared to laptop or desktop processors. Our benchmarking shows that AUIT can run consistently at 60fps in a scene with three UI elements (each with four adaptation objectives) without noticeable frame rate drops.

4.8 Using and Extending AUIT in a Project

Creators can add AUIT to an existing Unity project by importing the Unity package we make available. To design adaptations, a creator associates toolkit components to UI elements in the scene - by dragging and dropping or typing their names in the Unity inspector. Creators can customize AUIT components through the Unity inspector without requiring coding. Once adaptation objectives and property transitions are added, alongside a solver and respective adaptation trigger, AUIT is ready to adapt the UI. Creators can immediately visualize the adaptations they create by entering play mode in Unity. It is possible to adjust the weights of adaptation objectives in real-time (see Fig. 6) through the Unity inspector or in an immersive setting, allowing adaptive UIs to be experienced immediately through an XR device or simulation. To optimize multiple UI elements in the same optimization loop, creators must add these to an adaptation manager component and enable the option to use a global solver.

5 TOOLKIT EVALUATION

To evaluate the utility of AUIT, we conducted a study where creators of XR applications design adaptive UIs for two scenarios using AUIT. Assessing toolkit usage has been identified as a valuable step to evaluate what toolkits can do, whom they can support, and which tasks their users can perform [35]. The study aims to explore three research questions: (1) the *conceptual clarity of toolkit components*, (2) *toolkit usability*, and (3) the *quality of adaptations*.

5.1 Study design

First, the experimenter introduces participants to the design concepts present in AUIT. Then, they use AUIT to create adaptive user interfaces for two XR scenarios. To facilitate the introduction to the wide variety of features and customization the toolkit supports, the experimenter assists the creator throughout the study by answering questions about specifics of parameters and other technical details they find unclear. For that reason, we consider our study a combination of a usability study and a walkthrough demonstration, according to the toolkit evaluation methods identified by Ledo et al. [35]. Initially we planned to use MRTK's solvers [43] as a baseline. However, in a pilot study, we noticed how sequential optimization was insufficient to successfully fulfill the goals of our scenarios in a satisfactory manner. Such a baseline would require coding and result in an unfair comparison. For these reasons, we only evaluated AUIT.

The two scenarios present in our study are a video call (Figure 7a) and an interactive recipe (Figure 7b). To increase control and

(a) *Scenario 1: Video Call* (b) *Scenario 2: Interactive Recipe*

Figure 7: The UI panels from the two scenarios.

streamline the study, participants start each design task from a scene containing the corresponding UI elements in Figure 7.

To allow quick prototyping, we use a simulation of an MR application in Unity based on VirtualHome [58], to model activities occurring in a kitchen. Participants could quickly see the adaptations they create by starting the simulation, which shows what an end-user would see from a first-person point of view when using the application. The user randomly performs different tasks in the kitchen, such as opening the fridge to gather ingredients, moving to the stove or counter to prepare food, or having a break doing something else. We asked participants to consider the kitchen in our simulation as the real-world in an MR application, while the UIs from the scenarios would be the holograms the user sees through the HMD.

5.2 Participants

We recruited 8 experts for the study who develop XR applications and have experience with Unity (1 female, age: $M = 32.5, SD = 3.38$). Although there is no meaningful cut-off point for which the sample size is enough [7], note that the goal of our study was to gather qualitative feedback - it is not our intention to draw statistically significant conclusions. Four participants shared an academic background and conducted research in XR, while the remaining four reported jobs in the industry where they actively develop XR applications. On average, participants had several years of experience developing XR applications ($M = 4.5, SD = 2.56$). On a scale from 1 (low) to 5 (high), participants reported familiarity with Adaptive User Interfaces ($M = 3.25, SD = 0.71$) and some regularity in developing them ($M = 2.5, SD = 1.51$).

5.3 Procedure

After an introduction to the study and signing the consent form, participants went through the following phases:

5.3.1 Adaptive user interfaces and existing tools. We start the study with a discussion about adaptive user interfaces for XR experiences. Participants reported their opinion on their importance, and those who develop adaptations revealed their current methods and practices to implement them.

5.3.2 Introduction to AUIT. After introducing the concept of adaptive UIs and the aim of the study, the experimenter explained the toolkit components to the participant. Then, six adaptation objectives are showcased through videos, followed by a discussion about breakdowns that might occur when using naive strategies to combine multiple adaptation objectives. Participants reported how they currently handle adaptations with conflicting objectives or how they would do it if faced with such a problem.

5.3.3 Creating adaptive user interfaces using AUIT. Next, we start the main task, where participants use AUIT to create adaptive UIs for two scenarios:

Video call A video call application in XR, with a UI element (Figure 7a) that contains the video feed and basic call controls that support hand input. The end-user is performing tasks in the kitchen while on a video call. The goal is to create an adaptation policy where the video call is visible, in the user's reach, and the virtual element is not occluded.

Interactive Recipe A cooking application to provide recipe instructions, interactive videos, and access to the list of ingredients. In this scenario, the application contains two UI elements: 1) the instruction panel and 2) a panel with controls that support hand input (Figure 7b). The user interacts with the system through the control panel, where it is possible to change instructions, control the video, and spawn co-located timers in the kitchen. The goal is to create an adaptation policy where both UIs are visible and do not overlap with each other. In this scenario, only the control panel needs to be reachable.

The order of the tasks represents a learning curve, with a simple video call task first (one UI), followed by the interactive recipe scenario (two UIs). Participants are encouraged to meet other usability criteria they deem relevant. After each task we elicit user feedback by asking them to fill out a questionnaire and open-ended questions.

5.3.4 Feedback and discussion. Finally, we conduct an open-ended interview with the participants on aspects of AUIT, such as whether and how they would use it in their work and potential trade-offs of doing so.

5.4 Results

We started the study with a discussion about adaptive UIs for XR. All the participants considered adaptations to be crucial to provide a good user experience in XR applications: "It's really important because you want the user to [be able to] interact with the system when he moves around" (P4); "for XR [experiences] to not be frustrating [...] the interface [should] adapt to the environment" (P1). Creators also mentioned difficulties caused by the lack of resources and tools: "Resources to develop AUIs are limited" (P5); "I don't think [our company] has prioritized creating [AUIs], I think partly due to the [low] availability of tools to make it happen" (P8). Participants that actively develop adaptive user interfaces described their current practices are mostly based on custom rule-based implementations: "we develop our adaptations [...] we try to create our behaviors" (P4); "[we had to] customize MRTK behavior because often it doesn't behave the way we want [...] we faced [usability] issues and have solved them by implementing our own [rule-based] logic" (P6).

Table 1: The top row shows some of the potential usability breakdowns that can occur when using naive adaptation approaches in scenario 1; The bottom row shows adaptations created by a study participant during scenario 1 where AUIT overcomes some of these issues.

Table 2: Adaptations by participants for scenario 2 (top row - P3); (bottom row - P8). Both solutions fulfill visibility requirements while considering world geometry using different combinations of adaptation objectives, highlighting the toolkit's flexibility. While P3 attempted to group both UI elements, P8 focused on maintaining these in the same zones of the user's FoV over time.

All participants successfully designed adaptations that met the requirements of both scenarios. In *Scenario 1: Video Call*, the participants spent between 6 and 18.3 minutes ($M = 11.92, SD = 4.57$), whereas in *Scenario 2: Interactive Recipe* they took between 4.82 and 23.87 minutes ($M = 10.87, SD = 6.26$).

5.4.1 Conceptual clarity of design concepts. After the presentation of the toolkit, participants showed understanding of the different design concepts throughout the study, suggesting it provides a clear separation of concerns for the problem: "the adaptation speed is still slow, [but] that is outside of the scope of the adaptation objective." (P4); "In VR I would use the player's transform as the context [source] for the distance [interval] objective" (P5); "another [adaptation] objective that could be added [... is to] take the [user's FOV], make it into a grid, and then specify which cells you want [virtual objects] to be at" (P6). Moreover, a participant suggested the design concepts in AUIT make development easier: "[the toolkit components] are very much in line with the Unity philosophy of structuring behavior, which would make it easy to implement [XR adaptations]" (P8).

5.4.2 Toolkit usability. After each scenario, participants filled in a questionnaire with questions concerning the difficulty in using the toolkit to create adaptations for the video call (S1) and interactive recipe (**S2**) scenarios. On a scale from 1 (very easy) to 5 (very hard) participants reported that combining objectives was easy, but the difficulty slightly increases with more UI elements (**S1:** $M = 1.25, SD = 0.43$; **S2:** $M = 2.25, SD = 0.66$). Meanwhile, configuring components in the toolkit was reported as easy in both scenarios (**S1:** $M = 2.13, SD = 0.59$; **S2:** $M = 1.88, SD = 0.59$). In regards to finding appropriate weights for each objective, participants rated the difficulty of this task as medium for both scenarios (**S1:** $M = 2.5, SD = 0.86$; **S2:** $M = 2.75, SD = 0.18$). Note that participants used around 4 adaptation objectives in **S1** ($M = 4.13, SD = 0.59$) and 8 in **S2** ($M = 7.5, SD = 1.32$).

To develop adaptations using AUIT, creators followed a similar workflow throughout the study. They would add or remove components to the UI, such as adaptation objectives, tweak parameters, and run the simulation to visualize the resulting adaptations. Then, they would repeat this process until they were satisfied with the result, appreciating that they could see the adaptations created. A participant suggested the possibility to add commonly used combinations of toolkit components through a pre-configured and reusable asset to make the current workflow faster - "there are some objectives that often will go together [...] I like the [flexibility] to freely define what you want your behavior to be like [...] but it [requires some setup]" (P4).

Throughout the study, participants would occasionally ask for details about some of the parameters in components of the toolkit - "[Without documentation], it's hard to understand what some of these [parameters] do" (P5), highlighting the importance of resources to support the development and help developers get started with the toolkit. Nonetheless, they immediately understood many existing features due to familiarity with Unity and computer graphics concepts. As AUIT evolves, it is crucial to provide good tutorials and up-to-date documentation. In a few instances, experimenters helped participants debug specific behaviors in adaptations. Going forward, it is important to expand on the existing debugging capabilities of the toolkit, which are limited to console logging.

5.4.3 Quality of adaptations. When asked if they succeeded in creating the adaptations they had initially envisioned, participants reported on a scale from 1 (no, not at all) to 5 (yes, absolutely) that AUIT enabled them to do so in both scenarios (**S1:** $M = 4.38, SD = 0.48$; **S2:** $M = 3.75, SD = 0.82$). This feedback is interesting, considering that participants solved both tasks using different combinations of adaptation objectives, different settings for property transitions, and different parameters for the adaptation trigger. Even though all participants met basic usability requirements, such as visibility and reachability, some went a step further and considered factors such as consistency (about the position of the UI in relation

to the user) and grouping of related virtual elements (in scenario 2) - see Tables 1 and 2.

On a scale from 1 (very poor) to 5 (very high), participants rated the adaptations they created to be of high quality (**S1:** $M = 4, SD = 0.5$; **S2:** $M = 4.13, SD = 0.60$). When considering the time spent and the results obtained, creators reported using a scale from 1 (very inefficient) to 5 (very efficient), that they felt highly efficient (**S1:** $M = 4.38, SD = 0.48$; **S2:** $M = 4.25, SD = 0.66$). Participants also mentioned how the toolkit could be valuable in their current work, pointing out how it could make development faster: "This is a challenge that we have every time we create a HoloLens application. Our workflow is pretty much the same [all the time]. We use MRTK and then code until it behaves like we want to, which can take a lot of time" (P6). A participant with less coding expertise appreciated that it was easy to get started with AUIT: "I think it's a huge benefit, particularly from my standpoint where I don't do a lot of coding. Instead of creating a lot of scripts myself, this is a toolkit that would allow me to prototype things very quickly, test things out, and demonstrate [them to my team]" (P7).

6 DISCUSSION

We present AUIT, a toolkit to facilitate the design of adaptive UIs for XR applications. AUIT allows creators to combine adaptation objectives and find appropriate solutions using multi-objective optimization. Our approach lowers the barrier for creators to develop and experiment with adaptation policies while making the development process efficient through a clear separation of concerns. AUIT is a unifying toolkit that can bring together different adaptation methods, such as the variety of objectives we already support and other concepts explored in related work.

In our expert evaluation, participants pointed out that adaptive UIs are particularly important for providing good user experiences in XR applications. Moreover, they reported that existing tools lack appropriate support for designing adaptive UIs, requiring repetitive and time-consuming programming tasks. Their input suggests the toolkit components are understandable, and its separation of concerns can facilitate the development of adaptations.

Our study demonstrated how AUIT simplifies combining adaptation objectives and customizing adaptations, allowing creators to finish tasks involving complex behaviors in a short time while reporting feeling efficient doing so. All the participants successfully performed the proposed tasks, and their workflow showcases how AUIT can enable creative exploration of several aspects of adaptive UIs. Participants also reported that the toolkit allowed them to create the adaptations they had envisioned and rated their creations to be of high quality, suggesting how AUIT can support the creation of adaptive UIs.

AUIT is a plugin for Unity, so it can be used alongside other tools such as MRTK, giving creators more options without requiring drastic changes to existing workflows. Furthermore, it allows for quick prototyping of adaptive UIs - most game engines allow creators to visualize changes throughout development (e.g., play mode in Unity) - AUIT allows similar prototyping, letting users update several aspects of an adaptation without requiring scripting. Ideally, approaches such as our toolkit will be better integrated into XR development tools, reducing the challenges for creators to get started with the development of adaptations. Moreover, adding new components to AUIT to consider other adaptation aspects opens new opportunities to explore the immense design space of adaptive UIs.

6.1 Limitations and Future Work

AUIT presents a first step towards a general toolkit for the design of adaptive UIs for XR applications. In this first version we have focused on two fundamental usability issues of XR applications: visibility and reachability of individual UI elements. However, the five design concepts described in this paper allow to extend AUIT in the future, for example to address other usability issues, enable joint optimization of multiple UI elements or allow creators to interactively add constraints to the optimization. Now we discuss these opportunities for future work in more detail and underline limitations of our research.

Adaptation objectives. We implemented seven adaptation objectives related to fundamental usability issues in XR applications, but many others can contribute to better usability. Creators can build on existing research to implement new adaptation objectives as part of AUIT, such as ergonomics [16] or surface magnetism [70]. AUIT is currently limited to optimizing properties such as position, rotation, and scale, which are critical to ensure the visibility of UI elements. However, there are other properties of interest to adapt in XR settings, such as the level of detail or the decision to show or hide a UI element. Integrating those into AUIT is possible, but not straightforward, and requires more research to extend our solvers and support the consideration of other factors, such as the utility of a UI configuration and how it affects the user's cognitive load in our cost formulation. In this initial version of AUIT, we limit the optimization of multiple virtual elements to objectives that only consider properties of a single UI element at a time. An interesting direction is to extend AUIT to support global objectives that consider multiple UI elements, for example to avoid clutter in the user's FoV.

Solvers and alternative optimization methods. In the current implementation of our toolkit, we use a weighted sum method to combine multiple objectives into one cost function. This cost function is a linear combination of the objective's cost functions that allows a designer to set the weights according to the importance of individual adaptation objectives. This method works well if the particular costs of objectives have a similar scale, the reason why we designed normalized cost functions. However, an initial selection of weights does not guarantee the desired solution, requiring creators to adjust these during the design process. An additional disadvantage of using a weighted sum method is that it cannot find certain Pareto optimal solutions in the case of a non-convex objective space [41], a problem that other methods such as evolutionary algorithms could overcome [71].

Context widgets. We use context widgets that are already part of most XR development tools. Adding methods to retrieve context in high abstraction levels can enable the design of new adaptation objectives. Some options with potential are enhanced scene understanding [24, 28], measuring cognitive load [14], or full-body tracking.

Adaptation triggers. Although UI adaptations have the potential to improve usability [37], they can also be counterproductive [19, 65]. For example, adaptations can affect the user's attention or memory of the UI layout. One of the adaptation triggers implemented as part of our toolkit considers that adaptations come at a cost and only adapt when there are considerable improvements. However, other approaches could assess the utility of an adaptation to the user or add support for adding rules before or after running the optimization procedure.

Property transitions. Property transitions can avoid detrimental effects when adaptations occur and are currently under-explored in AUIT. Some examples of property transitions that could enhance usability are the replication of UI elements temporarily - to avoid changes in the layout - or anchor UI elements to body parts, such as the hands - a technique commonly used in applications supporting hand input.

Evaluation. Although our study suggests how AUIT can be relevant for creators, it has a small sample size of 8 experts. Replicating the study with more participants using a standardized test for usability, such as the System Usability Scale [6], could provide relevant quantitative results.

Other scenarios. Although we cover video calls and interactive cooking scenarios in our work, AUIT can generalize to other settings. An example would be sketching in 3D, whether in VR for creating 3D models, or AR for drawing in 3D. Here, a UI could adapt the position of a color palette to be easily reachable by the non-dominant hand. Another example is manufacturing, where workers often need to assemble separate parts into larger machinery. As the hands are busy, an AR interface can provide clear instructions to the user. The UI could adapt to be close to relevant parts of the user's assembling steps while avoid occlusion. Furthermore, it is interesting to consider AR in everyday life as a personal computing device. Users might frequently transition between different locations, rooms, and activities, bringing in a new level of complexity - tools such as AUIT can support the development of adaptive AR content to fit the user's context.

7 CONCLUSION

We presented AUIT, a toolkit to facilitate the design of adaptive UIs. AUIT implements five design concepts to support the development of such adaptations. The toolkit allows creators to combine multiple adaptation objectives as part of their development process and easily customize each aspect of an adaptive user interface. Our study with experts suggests that the design concepts we propose give creators a valuable separation of concerns for creating adaptations. Furthermore, AUIT allowed participants to efficiently design adaptive user interfaces that they rated to be of high quality. By making our toolkit widely available, we hope not only to lower the barrier for practitioners to get started creating adaptive UIs, but also to enable new workflows that allow for more creativity and require less repetitive and tedious tasks.

ACKNOWLEDGMENTS

This work was supported by the Innovation Fund Denmark (IFD grant no. 6151-00006B) as part of the Manufacturing Academy of Denmark (MADE) Digital project. Antti Oulasvirta was supported by the Finnish Center for Artificial Intelligence (FCAI), and Academy of Finland grants 'Human Automata' and 'BAD'. Special thanks to Aïna Linn Georges for the help with revisions and the anonymous reviewers for constructive feedback that helped improve the paper.

REFERENCES

[1] Khalil Amine. 2019. Multiobjective Simulated Annealing: Principles and Algorithm Variants. *Advances in Operations Research* 2019 (05 2019), 1–13. https://doi.org/10.1155/2019/8134674

[2] Narges Ashtari, Andrea Bunt, Joanna McGrenere, Michael Nebeling, and Parmit K. Chilana. 2020. *Creating Augmented and Virtual Reality Applications: Current Practices, Challenges, and Opportunities.* Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376722

[3] Blaine Bell, Steven Feiner, and Tobias Höllerer. 2001. View Management for Virtual and Augmented Reality. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* (Orlando, Florida) *(UIST '01).* Association for Computing Machinery, New York, NY, USA, 101–110. https://doi.org/10.1145/502348.502363

[4] Leonardo Bonanni, Chia-Hsun Lee, and Ted Selker. 2005. A Framework for Designing Intelligent Task-Oriented Augmented Reality User Interfaces. In *Proceedings of the 10th International Conference on Intelligent User Interfaces* (San Diego, California, USA) *(IUI '05).* Association for Computing Machinery, New York, NY, USA, 317–319. https://doi.org/10.1145/1040830.1040913

[5] Amani Braham, F. Buendia, Maha Khemaja, and Faiez Gargouri. 2021. User interface design patterns and ontology models for adaptive mobile applications. *Personal and Ubiquitous Computing* (01 2021), 1–17. https://doi.org/10.1007/s00779-020-01481-5

[6] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

[7] Kelly Caine. 2016. Local Standards for Sample Size at CHI. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16).* Association for Computing Machinery, New York, NY, USA, 981–992. https://doi.org/10.1145/2858036.2858498

[8] Eduardo Castillejo, Aitor Almeida, and Diego López de Ipiña. 2014. Ontology-Based Model for Supporting Dynamic and Adaptive User Interfaces. *International Journal of Human–Computer Interaction* 30, 10 (2014), 771–786. https://doi.org/10.1080/10447318.2014.927287 arXiv:https://doi.org/10.1080/10447318.2014.927287

[9] Yi Fei Cheng, Yukang Yan, Xin Yi, Yuanchun Shi, and David Lindlbauer. 2021. SemanticAdapt: Optimization-based Adaptation of Mixed Reality Layouts Leveraging Virtual-Physical Semantic Connections *(UIST '2021).* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3472749.3474750

[10] Perkins Coie. 2021. *2021 XR Survey.* https://www.perkinscoie.com/en/ar-vr-survey-results/2021-augmented-and-virtual-reality-survey-results.html

[11] Tilman Deuschel and Ted Scully. 2016. On the Importance of Spatial Perception for the Design of Adaptive User Interfaces. In *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO).* 70–79. https://doi.org/10.1109/SASO.2016.13

[12] Anind Dey, Gregory Abowd, and Daniel Salber. 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction* 16 (04 2001). https://doi.org/10.1207/S15327051HCI16234_02

[13] Edsger W Dijkstra. 1982. On the role of scientific thought. In *Selected writings on computing: a personal perspective.* Springer, 60–66.

[14] Andrew T. Duchowski, Krzysztof Krejtz, Izabela Krejtz, Cezary Biele, Anna Niedzielska, Peter Kiefer, Martin Raubal, and Ioannis Giannopoulos. 2018. The Index of Pupillary Activity: Measuring Cognitive Load Vis-à-Vis Task Difficulty with Pupil Oscillation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18).* Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173856

[15] 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).* Association for Computing Machinery, New York, NY, USA, 65–68. https://doi.org/10.1145/2788940.2788954

[16] João Marcelo Evangelista Belo, Anna Maria Feit, Tiare Feuchtner, and Kaj Grønbæk. 2021. XRgonomics: Facilitating the Creation of Ergonomic 3D Interfaces. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* Association for Computing Machinery, New York, NY, USA, Article 290, 11 pages. https://doi.org/10.1145/3411764.3445349

[17] Anna Maria Feit, Myroslav Bachynskyi, and Srinath Sridhar. 2015. Towards Multi-Objective Optimization for UI Design. (April 2015). http://annafeit.de/resources/papers/Multiobjective_Optimization2015.pdf Presented at CHI 2015

Workshop on Principles, Techniques and Perspectives on Optimization and HCI.

[18] Anna Maria Feit, Lukas Vordemann, Seonwook Park, Caterina Berube, and Otmar Hilliges. 2020. Detecting Relevance during Decision-Making from Eye Movements for UI Adaptation. In *ACM Symposium on Eye Tracking Research and Applications* (Stuttgart, Germany) *(ETRA '20 Full Papers)*. Association for Computing Machinery, New York, NY, USA, Article 10, 11 pages. https://doi.org/10.1145/3379155.3391321

[19] Leah Findlater and Krzysztof Z. Gajos. 2009. Design space and evaluation challenges of adaptive graphical user interfaces. *AI Magazine* 30, 4 (2009), 68–73. https://doi.org/10.1609/aimag.v30i4.2268

[20] Leah Findlater and Joanna McGrenere. 2008. Impact of Screen Size on Performance, Awareness, and User Satisfaction with Adaptive Graphical User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. Association for Computing Machinery, New York, NY, USA, 1247–1256. https://doi.org/10.1145/1357054.1357249

[21] Krzysztof Gajos and Daniel S. Weld. 2004. SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces* (Funchal, Madeira, Portugal) *(IUI '04)*. Association for Computing Machinery, New York, NY, USA, 93–100. https://doi.org/10.1145/964442.964461

[22] Krzysztof Gajos and Daniel S. Weld. 2005. Preference Elicitation for Interface Optimization. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology* (Seattle, WA, USA) *(UIST '05)*. Association for Computing Machinery, New York, NY, USA, 173–182. https://doi.org/10.1145/1095034.1095063

[23] Ran Gal, Lior Shapira, Eyal Ofek, and Pushmeet Kohli. 2014. FLARE: Fast layout for augmented reality applications. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 207–212. https://doi.org/10.1109/ISMAR.2014.6948429

[24] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. 2019. Mesh R-CNN. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[25] Camille Gobert, Kashyap Todi, Gilles Bailly, and Antti Oulasvirta. 2019. SAM: A Modular Framework for Self-Adapting Web Menus. In *Proceedings of the 24th International Conference on Intelligent User Interfaces* (Marina del Ray, California) *(IUI '19)*. Association for Computing Machinery, New York, NY, USA, 481–484. https://doi.org/10.1145/3301275.3302314

[26] Raphaël Grasset, Tobias Langlotz, Denis Kalkofen, Markus Tatzgern, and Dieter Schmalstieg. 2012. Image-driven view management for augmented reality browsers. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 177–186. https://doi.org/10.1109/ISMAR.2012.6402555

[27] Uwe Gruenefeld, Jonas Auda, Florian Mathis, Stefan Schneegass, Mohamed Khamis, Jan Gugenheimer, and Sven Mayer. 2022. VRception: Rapid Prototyping of Cross-Reality Systems in Virtual Reality. (2022).

[28] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[29] Otmar Hilliges, Christian Sandor, and Gudrun Klinker. 2006. Interactive Prototyping for Ubiquitous Augmented Reality User Interfaces. In *Proceedings of the 11th International Conference on Intelligent User Interfaces* (Sydney, Australia) *(IUI '06)*. Association for Computing Machinery, New York, NY, USA, 285–287. https://doi.org/10.1145/1111449.1111512

[30] Juan David Hincapié-Ramos, Xiang Guo, Paymahn Moghadasian, and Pourang Irani. 2014. Consumed Endurance: A Metric to Quantify Arm Fatigue of Mid-Air Interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 1063–1072. https://doi.org/10.1145/2556288.2557130

[31] Tobias Hans Höllerer. 2004. *User Interfaces for Mobile Augmented Reality Systems*. Ph. D. Dissertation. Columbia University.

[32] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. 1983. Optimization by Simulated Annealing. *Science (New York, N.Y.)* 220 (06 1983), 671–80. https://doi.org/10.1126/science.220.4598.671

[33] Sarah Krings, Enes Yigitbas, Ivan Jovanovikj, Stefan Sauer, and Gregor Engels. 2020. Development Framework for Context-Aware Augmented Reality Applications. In *Companion Proceedings of the 12th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Sophia Antipolis, France) *(EICS '20 Companion)*. Association for Computing Machinery, New York, NY, USA, Article 9, 6 pages. https://doi.org/10.1145/3393672.3398640

[34] Wallace S. Lages and Doug A. Bowman. 2019. Walking with Adaptive Augmented Reality Workspaces: Design and Usage Patterns. In *Proceedings of the 24th International Conference on Intelligent User Interfaces* (Marina del Ray, California) *(IUI '19)*. Association for Computing Machinery, New York, NY, USA, 356–366. https://doi.org/10.1145/3301275.3302278

[35] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. *Evaluation Strategies for HCI Toolkit Research*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3173574.3173610

[36] Germán Leiva, Jens Emil Grønbæk, Clemens Nylandsted Klokmose, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2021. Rapido: Prototyping Interactive AR Experiences through Programming by Demonstration. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 626–637. https://doi.org/10.1145/3472749.3474774

[37] David Lindlbauer, Anna Maria Feit, and Otmar Hilliges. 2019. Context-aware online adaptation of mixed reality interfaces. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 147–160.

[38] Feiyu Lu, Shakiba Davari, Lee Lisle, Yuan Li, and Doug A Bowman. 2020. Glanceable ar: Evaluating information access methods for head-worn augmented reality. In *2020 IEEE conference on virtual reality and 3D user interfaces (VR)*. IEEE, 930–939.

[39] Feiyu Lu and Yan Xu. 2022. Exploring Spatial UI Transition Mechanisms with Head-Worn Augmented Reality. In *ACM CHI Conference on Human Factors in Computing Systems (CHI '22)*. ACM.

[40] Jacob Boesen Madsen, Markus Tatzqern, Claus B. Madsen, Dieter Schmalstieg, and Denis Kalkofen. 2016. Temporal Coherence Strategies for Augmented Reality Labeling. *IEEE Transactions on Visualization and Computer Graphics* 22, 4 (2016), 1415–1423. https://doi.org/10.1109/TVCG.2016.2518318

[41] R Timothy Marler and Jasbir S Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization* 26, 6 (2004), 369–395.

[42] Microsoft. 2022. *About Hololens 2*. https://docs.microsoft.com/en-us/hololens/hololens2-hardware

[43] Microsoft. 2022. *Solver overview — MRTK2*. https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/solvers/solver?view=mrtkunity-2022-05

[44] Microsoft. 2022. *Spatial awareness getting started — MRTK2*. https://docs.microsoft.com/en-gb/windows/mixed-reality/mrtk-unity/mrtk2/features/spatial-awareness/spatial-awareness-getting-started?view=mrtkunity-2021-05

[45] Microsoft. n.d.. *Mixed Reality Toolkit (MRTK) for Unity*. https://github.com/microsoft/MixedRealityToolkit-Unity

[46] Mahdi H. Miraz, Maaruf Ali, and Peter S. Excell. 2021. Adaptive user interfaces and universal usability through plasticity of user interface design. *Computer Science Review* 40 (2021), 100363. https://doi.org/10.1016/j.cosrev.2021.100363

[47] G. Mori, F. Paterno, and C. Santoro. 2004. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering* 30, 8 (2004), 507–520. https://doi.org/10.1109/TSE.2004.40

[48] Leon Müller, Ken Pfeuffer, Jan Gugenheimer, Bastian Pfleging, Sarah Prange, and Florian Alt. 2021. SpatialProto: Exploring Real-World Motion Captures for Rapid Prototyping of Interactive Mixed Reality. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 363, 13 pages. https://doi.org/10.1145/3411764.3445560

[49] Michael Nebeling, Katy Lewis, Yu-Cheng Chang, Lihan Zhu, Michelle Chung, Piaoyang Wang, and Janet Nebeling. 2020. XRDirector: A Role-Based Collaborative Immersive Authoring System. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376637

[50] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. 2002. Generating Remote Control Interfaces for Complex Appliances. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology* (Paris, France) *(UIST '02)*. Association for Computing Machinery, New York, NY, USA, 161–170. https://doi.org/10.1145/571985.572008

[51] Lauren Norrie and Roderick Murray-Smith. 2016. Investigating UI Displacements in an Adaptive Mobile Homescreen. 8, 3 (2016). https://doi.org/10.4018/IJMHCI.2016070101.oa

[52] Benjamin Nuernberger, Eyal Ofek, Hrvoje Benko, and Andrew D. Wilson. 2016. SnapToReality: Aligning Augmented Reality to the Real World. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1233–1244. https://doi.org/10.1145/2858036.2858250

[53] Allan Oliveira and Regina B. Araujo. 2012. Creation and Visualization of Context Aware Augmented Reality Interfaces. In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (Capri Island, Italy) *(AVI '12)*. Association for Computing Machinery, New York, NY, USA, 324–327. https://doi.org/10.1145/2254556.2254618

[54] Antti Oulasvirta, Niraj Ramesh Dayama, Morteza Shiripour, Maximilian John, and Andreas Karrenbauer. 2020. Combinatorial Optimization of Graphical User Interface Designs. *Proc. IEEE* 108, 3 (2020), 434–464. https://doi.org/10.1109/JPROC.2020.2969687

[55] Seonwook Park, Christoph Gebhardt, Roman Rädle, Anna Feit, Hana Vrzakova, Niraj Dayama, Hui-Shyong Yeo, Clemens Klokmose, Aaron Quigley, Antti Oulasvirta, and Otmar Hilliges. 2018. AdaM: Adapting Multi-User Interfaces for Collaborative Environments in Real-Time. In *SIGCHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA.

[56] Fabio Paterno, Carmen Santoro, and Lucio Davide Spano. 2009. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented

Applications in Ubiquitous Environments. *ACM Trans. Comput.-Hum. Interact.* 16, 4, Article 19 (nov 2009), 30 pages. https://doi.org/10.1145/1614390.1614394

[57] Ken Pfeuffer, Yasmeen Abdrabou, Augusto Esteves, Radiah Rivu, Yomna Abdelrahman, Stefanie Meitner, Amr Saadi, and Florian Alt. 2021. ARtention: A design space for gaze-adaptive user interfaces in augmented reality. *Computers & Graphics* 95 (2021), 1–12.

[58] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. VirtualHome: Simulating Household Activities via Programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[59] Scott D Roth. 1982. Ray casting for modeling solids. *Computer graphics and image processing* 18, 2 (1982), 109–144.

[60] Paulo Salem. 2017. User Interface Optimization Using Genetic Programming with an Application to Landing Pages. *Proc. ACM Hum.-Comput. Interact.* 1, EICS, Article 13 (June 2017), 17 pages. https://doi.org/10.1145/3099583

[61] Christian Sandor and Gudrun Klinker. 2005. A Rapid Prototyping Software Infrastructure for User Interfaces in Ubiquitous Augmented Reality. *Personal Ubiquitous Computing* 9, 3 (2005), 169–185. https://doi.org/10.1007/s00779-004-0328-1

[62] Ryo Suzuki, Rubaiat Habib Kazi, Li-Yi Wei, Stephen DiVerdi, Wilmot Li, and Daniel Leithinger. 2020. RealitySketch: Embedding Responsive Graphics and Visualizations in AR with Dynamic Sketching. In *Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 135–138. https://doi.org/10.1145/3379350.3416155

[63] Markus Tatzgern, Denis Kalkofen, Raphael Grasset, and Dieter Schmalstieg. 2014. Hedgehog labeling: View management techniques for external labels in 3D space. In *2014 IEEE Virtual Reality (VR)*. 27–32. https://doi.org/10.1109/VR.2014.6802046

[64] Markus Tatzgern, Valeria Orso, Denis Kalkofen, Giulio Jacucci, Luciano Gamberini, and Dieter Schmalstieg. 2016. Adaptive information density for augmented reality displays. In *2016 IEEE Virtual Reality, VR 2016, Greenville, SC, USA, March 19-23, 2016*, Tobias Höllerer, Victoria Interrante, Anatole Lécuyer, and Evan A. Suma (Eds.). IEEE Computer Society, 83–92. https://doi.org/10.1109/VR.2016.7504691

[65] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting User Interfaces with Model-Based Reinforcement Learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 573, 13 pages. https://doi.org/10.1145/3411764.3445497

[66] Unity. n.d.. *Unity Coroutines*. https://docs.unity3d.com/Manual/Coroutines.html

[67] Unity. n.d.. *Unity Mars*. https://unity.com/products/unity-mars

[68] V. Černý. 1985. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. 45, 1 (Jan. 1985), 41–51. https://doi.org/10.1007/BF00940812

[69] Robert Xiao, Scott Hudson, and Chris Harrison. 2017. Supporting Responsive Cohabitation Between Virtual Interfaces and Physical Objects on Everyday Surfaces. *Proc. ACM Hum.-Comput. Interact.* 1, EICS, Article 12 (jun 2017), 17 pages. https://doi.org/10.1145/3095814

[70] 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* 24, 4 (2018), 1653–1660. https://doi.org/10.1109/TVCG.2018.2794222

[71] E. Zitzler, Laumanns, M., and L. Thiele. 2001. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, EUROGEN*.

A IMPLEMENTATION DETAILS

In this section we describe how the cost functions and optimization heuristics for each adaptation objective are implemented. Note that the adaptation objectives are described in a different order here to optimize page space.

In this pseudo-code, variables declared before the functions are obtained dynamically or have default values that can be customized in the Unity inspector. We omit some software engineering technicalities - for more technical details please refer to the source code.

A.1 Adaptation Objectives

Algorithm 1: Distance Interval Objective

$gXZ \leftarrow$ goal distance from context source in XZ plane
$iXZ \leftarrow$ distance interval from gXZ (no cost penalty)
$iY \leftarrow$ height of the hollow cylinder (no cost penalty)
$t \leftarrow$ threshold for highest cost
$uiXZ \leftarrow$ UI XZ coordinates
$uiY \leftarrow$ UI Y coordinate
$csXZ \leftarrow$ context source XY coordinates
$csY \leftarrow$ context source Y coordinate
function Cost
  $difXZ \leftarrow csXZ - uiXZ$
  $dXZ \leftarrow$ magnitude($difXZ$) ▷ dist. from cs in XZ plane
  $dt \leftarrow$ abs($dXZ - gXZ$) ▷ UI XZ distance from goal
  $cXZ \leftarrow \max(0, dt - iXZ)$ ▷ no penalty if dist. $\leq iXZ$
  $dY \leftarrow$ abs($csY - uiY$) ▷ UI Y distance from goal
  $cY \leftarrow \max(0, dY - iY/2)$ ▷ no penalty if dist. $\leq iY/2$
  $c = cXY + cY$
  $c \leftarrow \min(c/t, 1)$ ▷ normalize cost according to t
  return c
end function
function Heuristics
  $s \leftarrow$ random value $\in [0, 1]$
  if $s \leq 0.5$ **then**
    $gXZ \leftarrow csXZ - uiXZ$
    $dXZ \leftarrow$ magnitude(gXZ) $- gXZ$ ▷ distance from goal
    $guXZ \leftarrow$ normalize(gXZ)
    $nPXZ \leftarrow uiXZ + guXZ * \mathcal{N}(dXZ, 0.2)$ ▷ move to goal
    $gY \leftarrow csY - uiY$
    $nPY \leftarrow uiY + gY * \mathcal{N}(0.3, 0.2)$ ▷ move to goal
    return nP ▷ new position likely closer to goal
  else
    $rU \leftarrow$ random unit vector
    return $uiPos + rU * \mathcal{N}(0.3, 0.2)$ ▷ move at random
  end if
end function

Algorithm 2: Avoid Occlusion Objective

$ks \leftarrow$ UI keypoint array dynamically generated (local coords.)
$csPos \leftarrow$ context source position
$uiTRS \leftarrow$ UI TRS matrix
$uiPos \leftarrow$ UI position
function Cost
  $c \leftarrow 0$
  for all k in ks **do**
    $wK = uiTRS \cdot k$ ▷ get k in world coord.
    if raycast($csPos, wK$) hits **then**
      $c \leftarrow c + 1$ ▷ increase cost
    end if
  end for
  return c/length(ks) ▷ return normalized cost
end function
function Heuristics
  $s \leftarrow$ random value $\in [0, 1]$
  if $s \leq 0.5$ **then** ▷ pick heuristic at random
    for all k in ks **do**
      $gP = uiTRS \cdot k$ ▷ get k in world coord.
      if raycast($csPos, wK$) hits **then**
        $n \leftarrow$ hit normal($csPos, wK$) ▷ surface normal
        return hit pos. $+ n * \mathcal{N}(1, 0.5)$ ▷ move away
      end if
    end for
  else
    $rU \leftarrow$ random unit vector
    return $uiPos + rU * \mathcal{N}(0.3, 0.2)$ ▷ move at random
  end if
end function

Algorithm 3: Anchor to Target Objective

$o \leftarrow$ offset vector provided by creator
$t \leftarrow$ threshold for highest cost
$csTRS \leftarrow$ context source TRS matrix
$uiPos \leftarrow$ UI position
function Cost
  $l \leftarrow csTRS^{-1} \cdot uiPos$ ▷ get UI position in cs local coord.
  $d \leftarrow$ distance(l, o) ▷ distance from offset to UI
  $c \leftarrow \min(d/t, 1)$ ▷ normalize cost according to t
  return c
end function
function Heuristics
  $s \leftarrow$ random value $\in [0, 1]$
  $opt \leftarrow csTRS^{-1} \cdot o$ ▷ compute optimal position
  if $s \leq 0.33$ **then** ▷ pick heuristic at random
    return opt ▷ return optimal position
  else
    $ou \leftarrow$ normalize ($opt - uiPos$)
    $uv \leftarrow$ random unit vector ▷ add randomness
    $ou \leftarrow ou +$ $uv *$ random value $\in [0, 0.3]$
    return $uiPos + ou * \mathcal{N}(1, 0.5)$
  end if
end function

Algorithm 4: Spatial Coherence Objective

```
u ← adaptations allowed until a position is forgotten
vs ← data structure for visited voxel data
uiPos ← position of the UI
function OnAdapt(pos)                          ▷ called whenever ui adapts
  for all v in vs
    decrease v score by 1
    if v score is 0 then
      remove v from vs
    end if
  end for
  add/update voxel at pos to vs with score u
end function
function Cost
  if vs has voxel containing uiPos then
    return 0
  else
    return 1
  end if
end function
function Heuristics
  s ← random value ∈ [0, 1]
  if s ≤ 0.5 then                              ▷ pick heuristic at random
    return voxel in vs closest to uiPos
  else
    return voxel in vs at random
  end if
end function
```

Algorithm 5: Constant View Size Objective

```
sF ← scaling factor                            ▷ scaling based on linear function
iS ← scaling difference tolerance (no cost penalty)
d ← base scale intended distance for UI
t ← threshold for highest cost
dS ← UI default scale
uiS ← UI current scale
uiP ← UI position
csP ← context source position
function Cost
  d ← magnitude(uiP − csP)                     ▷ get distance from ui to cs
  i ← dS ∗ (d/iS ∗ sF)                         ▷ ideal scale based on d
  c ← magnitude(uiS/i)                         ▷ compute scale difference
  return min(c/t, 1)                           ▷ normalize according to t
end function
function Heuristics
  s ← random value ∈ [0, 1]
  if s ≤ 0.5 then
    d ← magnitude(uiP − csP)                   ▷ dist. from ui to cs
    i ← dS ∗ (d/iS ∗ sF)                       ▷ optimal scale based on d
    return i ∗ N(1, 0.3)
  else
    r ← N(0.3, 0.2)
    return uiS ∗ r                             ▷ randomize scale
  end if
end function
```

Algorithm 6: Field of View Objective

```
bo ← boundary origin for desired region in FoV
i ← angle interval from bo (no cost penalty)
t ← angle threshold for highest cost
csTRS ← context source TRS matrix
uiPos ← UI position
function Cost
  l ← csTRS^−1 · uiPos                         ▷ get UI position in cs local coord.
  a ← angle([0, 0, 1], l)                      ▷ angle between gaze and UI
  a ← abs(a − bo)                              ▷ distance from desired origin
  a ← max(0, a)                                ▷ no penalty if angle ≤ i
  return min(a/t, 1)                           ▷ return normalized cost
end function
function Heuristics
  s ← random value ∈ [0, 1]
  lUI ← csTRS^−1 · uiPos                       ▷ get UI pos. in cs local coord.
  if s ≤ 0.5 then                              ▷ pick heuristic at random
    a ← angle ([0, 0, 1], l)                   ▷ angle between gaze and UI
    dir = 1                                    ▷ move towards cs forward
    if a − i ≤ 0 then
      dir = −1                                 ▷ move away from cs forward
    end if
    gW = csTRS · [0, 0, magnitude(lUI)]
    g ← gW − uiPos
    return uiPos + g ∗ dir ∗ N(0.3, 0.2)
  else
    rU ← random unit vector
    return uiPos + rU ∗ N(0.3, 0.2)            ▷ move at random
  end if
end function
```

Algorithm 7: Look Towards Objective

```
csPos ← context source position
uiZ ← UI "look" vector                         ▷ UI forward or equivalent
uiPos ← UI position
t ← angle threshold for highest cost
function Cost
  lA ← uiPos − csPos                           ▷ Vector from cs to ui
  a ← angle(lA, uiZ)                           ▷ angle difference
  return min(a/t, 1)                           ▷ return normalized cost
end function
function Heuristics
  lA ← uiPos − csPos                           ▷ vector pointing to target
  rot ← interpolate between uiZ and lA by N(1, 0.3)
  return rot as a Quaternion
end function
```