A VR End-User Development Toolbox for Media Study Students - An Initial Experience Report

Sebastian Krois[1], Kevin Scharke[1], Enes Yigitbas[2] and Gudrun Oevel[1]

[1]Paderborn University, Warburger Straße 100, 33098, Paderborn, Germany
[2]Paderborn University, Zukunftsmeile 2, 33102, Paderborn, Germany

Abstract

Virtual Reality (VR) has a wide range of possible applications, one of which is education. In a course of our University's media study, students create VR applications to be used in future iterations of courses they already passed, so younger students benifit from their experience. They learn VR development as well as think about concepts to integrate VR into teaching. Students may but not need to have previous development experience, so we need a tool that is usable for beginners but does not restrict experienced users. To achieve that, we develop a toolbox as an additional layer on Unity's XR Interaction Toolkit Examples. In this paper, we describe the concept, current development state, and results of an initial evaluation during the latest iteration of the course. The tool itself was, apart from smaller bugs, rather easy to use, but we identified the need to create (better) materials for introduction and guidance.

Keywords

Virtual Reality, End-User Development, VR Toolbox, VR Development in Unity

1. Introduction

Over the last years, Virtual Reality (VR) applications have become part of the multimedia landscape. But they are on the rise in educational and learning contexts as well [1]. Here, they are often used as a medium to convey some content to students. Another, less common, approach is to teach the development of a VR application. For a course in our universities' media studies, we combined both concepts. The course is a practical seminar for bachelor's students in media studies, but is also open for computer science students. In small groups, they create a VR application which can be used to support some other course they had to take during their studies. With that, they can reflect on their own study and select a topic they found difficult to understand. After completion, teachers can either use the application directly or use it as proof of concept and continue the further development. To enable students without prior experience to create such an application, we need a tool or development environment. In this paper, we first present the requirements we have identified for a tool to use in the course. In Section 2, we present some works and applications, including commercial as well as scientific projects. Following, in Section 3, we present our approach of creating a VR development toolbox, and describe our first experiences and learnings in Section 4. Finally, we summarize the project in Section 5.

Requirements We first define requirements the tool has to fulfill. It is meant to be mainly used by unexperienced developers and users who never developed an application before. We base our requirements on the paradigm of End-User Development (EUD) [2]. There are different approaches for VR-EUD editors (c.f. Section 2), which define requirements and create tools for creating VR applications. But the ones we found were e.g. specific to their domain and not tailored to be used by media study students. To achieve this for an arbitrary VR application in the educational context, we derive the following requirements.

Joint Proceedings of IS-EUD 2025: 10th International Symposium on End-User Development, 16-18 June 2025, Munich, Germany.
sebastian.krois@upb.de (S. Krois); kscharke@mail.upb.de (K. Scharke); enes.yigitbas@upb.de (E. Yigitbas); gudrun.oevel@upb.de (G. Oevel)
0009-0002-5116-9132 (S. Krois); 0009-0002-9002-1050 (K. Scharke); 0000-0002-5967-833X (E. Yigitbas); 0000-0002-6396-9535 (G. Oevel)

R1: Ease of use As the course is part of media studies, we cannot rely on the student's programming skills or knowledge about game/VR development in general. Hence, the tool needs to be easy enough to understand, so we only need a small amount of time for training on the tool. Additionally, users in EUD scenarios should be able to do the work themselves. So the application should not need additional work or support to be able to run.
R2: Free Choice of Assets We do not know in advance, what the application will be about. The students will develop the concept during the course. So we cannot provide an editor with predefined assets, but need to allow the creation of assets or to use some which are publicly available.
R3: Variety of Interaction By using VR, we can explore a wide range of interaction types. The tool should allow different types of simple interaction out-of-the-box but also offer the possibility for extensions with complex interactions.
R4: Create Logic Flow Not only should there be ways to interact with assets, but we need to configure effects for the user's actions. To be able to create meaningful gameplay that utilizes the features of VR, the tool needs an easy-to-use interface to define *what* happens *when*.
R5: Wide Platform Support The developed applications are aimed to be used in different courses at the university. As there exist different VR platforms, we want the applications to run on as many of them as possible.
R6: Ease of Initialization The tool's setup should be easy and do not require many steps to complete.

2. Related Work

The idea of creating end-user editors for VR is not new. There are multiple applications, created for research projects as well as commercially available ones. In this section, we give an overview of projects that aim to provide such an editor from both categories.

Research Projects In 2022, Coelho et al.[3] conducted a systematic literature review of VR authoring tools. Here, we see that most academic projects focus on the creation of 360° images or videos([4, 5]). Interacting with objects and the environment is a crucial part and one of the main advantages of using VR. So we cannot use projects aiming at creating only 360° views. Another category of works created tools, which are domain-specific. So, even when most of the requirements were fulfilled, they are tied to their domain [6, 7, 8] or are designed for other concepts going further than VR and thus need more setup [9, 7]. Also, there are tools, which need to be hosted on a web server [10]. They need to be set up by a developer and the server needs to be maintained so the tool is available when being used. That creates the need for a developer with a larger skillset than the users the application is created for. Some projects, like XRSpotlight by Frau et al.[11], are designed to help users to get started with XR, but they expect users to have some prior knowledge of programming.
Finally, the tool will be used in the lecture, so we need the project to be available and running on modern hardware. For most of the projects presented above, the project itself is not available or we were not able to set them up and get them running as they use out-of-date libraries and are not updated since publication.

Commercial In addition to tools emerging from research, there are some commercially available end-user editors for VR. Similar to the scientific projects (Section 2), there are multiple tools, allowing the creation of scenes containing 360° images or videos (Mobfish[1]). Some other projects (SimLab Soft VR Studio[2] or spatial.io[3]) offer the possibility to create interactive VR scenes. You can use predefined or your own models, and they support multiple platforms. But they come with only limited access to features in the free version and, even with paid subscriptions, offer only a small set of possible interactions. In previous iterations of the course, we used Spatial to create virtual environments. It is designed to be simple and only provide the most necessary controls for creating a VR environment, like grabbing an object to move it and scale or rotate it using gestures, users can also upload their own 3D models and other assets. However, the free version of Spatial only allows for 500MB of storage space and does not provide a way to completely delete models. Using the VR and web versions only provide grabbing. With

[1] https://mobfish.net [2] https://simlab-soft.com [3] https:/spatial.io

the Unity framework, a few further interactions are possible, but they are still very limited, and editing via VR or web is not possible. Finally, everything is stored on and loaded from a server, our students experienced technical difficulties, such as the server not loading the level or changes made not being saved.

Discussion As presented above, there are some applications for creating VR applications available. But for the academic projects, the applications are often not available anymore or are designed to only work in a very specific domain. The commercial applications are mostly available, but, even with a paid subscription, do not fulfill our requirements. Additionally, when tools rely on a server connection, connection errors will most likely occur after some time. To have a reliable tool available during the course, which satisfies our requirements, we created our own VR-Toolkit on top of Unity, which we currently use and evaluate. The tool is independent from the domain it is used for, so, when the project is published upon completion, it can be used for all kinds of domains, not just teaching.

3. Concept and Implementation

The requirements defined in Section 1 can be grouped into two types. First, there are requirements to ensure working with the toolkit is easy (**R1, R5, R6**, *simplicity*). Second, we ensure that users have great freedom to create the application they want in its correct domain (**R2, R3, R4**, *support creativity*).

Support Creativity To balance between simplicity and the support of creativity, we searched for a base to start, which does not restrict developers' possibilities. As we use the Unity game engine for multiple courses and lectures, we decided to also use it for the toolbox. To communicate with the headsets, we used Unity's XR Interaction Toolkit[4] with examples project[5]. It comes with a complete Character Controller and a wide range of interactions which are fully implemented and configured to work with their corresponding 3D models. To add 3D models (**R2**), Unity can import some model formats by default (e.g. fbx). When creating models by themselves, users can use one of the supported file formats. Modells from an asset library often use formats like .glb or .gltf. With adding glTFast[6], to the project, they can also be used. Users have full access to all of Unity's and the SDK's APIs and features (**R3**, **R4**), so experienced users or programmers are not restricted in what they want to create. Therefore we have a base, which fully supports the *support creativity* requirements.

Provide Simplicity By using Unity's XR Interaction Toolkit, the application can run on all major VR platforms which support OpenXR (**R5**). For the setup, users need to open Unity's XR Interaction Toolkit Examples and drag-and-drop a .unitypackage into the project. With that the toolbox is fully set-up (**R6**). To provide the desired simplicity (**R1**), we take the features provided and add a layer on top to simplify its use. We remove information and configuration options when they are not necessary. For the remaining information and options, we create a simplified visualization and extend the toolkit to support a simple, event-based logic flow.

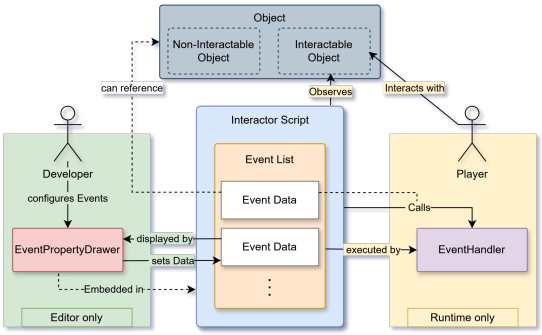

Figure 1: Architecture Overview

In Section 3.1 we describe the architecture we used to allow the logic flow. Followed by Section 3.2, where we explain the simplified inspector. Finally, in Section 3.3, we present the objects and functionalities, we made available in the toolkit. The most recent version of the toolkit is available here (https://github.com/ZIM-VR/VR-Toolkit) and will be updated until completion.

3.1. Architecture Overview

Figure 1 describes the applications' architectural overview for the editor and runtime of the toolbox. The virtual world consists of different *objects* (dark blue), which can be created and configured by

[4] docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1

[5] github.com/Unity-Technologies/XR-Interaction-Toolkit-Examples

[6] docs.unity3d.com/Packages/com.unity.cloud.gltfast@6.10

the developing user (*developer*). With some of them (*Interactable*), the player can interact, e.g. grab them. The others (*Non-Interactable*) can be for decoration only, or provide different functionalities. In Section 3.3, we describe the available objects in more detail. All objects can be created and configured (e.g. positioned) by the developer. To allow the creation of a logic flow, we created an event-based workflow. *Interactable Objects* have an *Interactor Script* (light blue) attached, which observes if its specific interaction happened. That can be the object being grabbed by the player or placed in a specific spot and triggering an associated event. Developers can configure what happens after the event occurs in the edit mode (green) using a custom interface (Section 3.2). For that, they can define a set of actions (*Event List*, orange), which are executed after the event occurs. For the action, we need the object, *where* it should happen and the configuration *what* should happen. Each pair of object and action is stored in a data holder called *Event Data* (white). When, during runtime (yellow), the *Interactor Script* observes interaction with the *Player*, it passes the corresponding *Event List* to the *Event Handler* (violet). It then iterates over all *Event Data* elements of the event list and executes the actions.

3.2. Simplified Inspector

As described above, we want to simplify the configuration as much as possible. For that, we decided to only show as much information as necessary to configure the object, and also let the users only configure what they really need to. As we want to stay inside Unity, we decided to override its default inspector by creating a custom *PropertyDrawer*. We implemented two types of custom inspector windows. The first is made for objects developers do not have to interact with, e.g. decorative objects. In that case, we override the existing inspector window with an empty one, so no data is displayed. If developers need access to an overridden and therefore hidden inspector nevertheless, we provide a separate settings window, where specific components can be whitelisted so they will be shown.

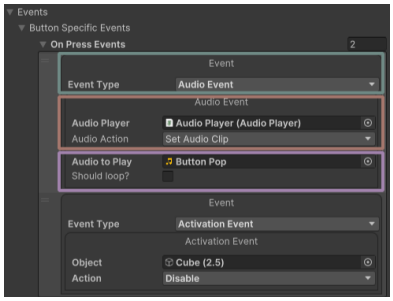

Figure 2: View in Unity

The second type of inspector is made for objects the developer configures (Figure 2). The inspectors for interactable objects follow the same layout, which consists of multiple fields describing an action. In that, events can be configured and will be executed as soon as the action triggers. The event field consists of three parts. The upper section (green) consists of a field where the event's type can be specified i.e. which kind of action is executed when the event triggers. These currently include *activation* (turn an object on or off), *audio* (play/stop audio), *video* (play/stop a video), *light*, and *scene* (load a scene), c.f. Section 3.3. Additionally, it is possible to select the type *Unity Event* which shows the default *UnityEvent* user interface. This allows experienced developers to create and execute their own actions. Depending on the event's type, the *PropertyDrawer* displays different user interfaces which all follow the same visual hierarchy. For actions that are executed on a specific object, it can be referenced at the top. For some types (e.g. *scene*), the execution is not bound to one object, so we do not need to configure one. To provide simplicity, the object field only accepts objects of the corresponding types, e.g., an audio event's object field would only accept an *Audio Player* object. Below, there is a drop-down field where users can choose one of the actions available (red) on the selected object. When an action is selected, further fields for configuration are displayed (violet). Selecting *'stop audio'* does not need additional parameters unlike the event action *'set audio clip'* which needs an audio clip field.

3.3. Available Prefabs

In previous semesters, we asked students, which kind of interaction and functionalities they would like to have available for creating VR applications. Based on their suggestions and on the tools provided by Unity's *Interaction Toolkit Examples* we created the following prefabs.

Interactable An example of an interactable object is the *PushButton* — a button that can be pressed in the virtual reality environment. One of the available actions is *'OnButtonPress'* which is triggered when the button is pushed in, or *'OnButtonRelease'* which triggers when the button is released. Here,

developers can configure the actions as described in Section 3.2. Other interactable objects are *Grabbable Objects*, *Socket Interactors* (grabbable objects which can be placed on) *Socket Shapes*, and *Player Triggers* (areas that invoke an event when the player enters or exits it).

Non-Interactable Players cannot interact with such objects, but they can still execute an action. In most cases, the action is not configured on the Non-Interactable object itself, but within the Event Data of the Interactable object invoking the event. A Non-Interactable object is, for example, the *AudioPlayer*. Players cannot directly interact with it — but can be configured, e.g., to play a sound after a *PushButton* is pressed. The *AudioPlayer* could also be used to play background music. In this case, it would not play an audio file after an event occurred, but is configured to play from the application startup. Other non-interactable objects are *PlayerSpawnpoints* (determines, where the player starts), *TextFields*, and *VideoPlayers*. All objects presented above use a simplified inspector (Section 3.2). As we build upon Unity's tools, experienced users can create their own (Non-) Interactable objects or commission that feature from other developers. Those objects can then be used together with the ones we already provide. Finally, as described at the beginning of this section, we support different forms of custom-imported 3D models. All functionalities described above can easily be used with custom models. To do so, the custom models can be drag-and-dropped into the corresponding prefab, if necessary the scale and position of the model can be adjusted.

4. Initial Evaluation

For an initial evaluation, we let our students use the work-in-progress version of our tool in the latest iteration of the course. 12 students were divided into five groups, each created a VR application. To give an example of how the tool was used, we briefly describe one project. The application lets player listen to music using different player mediums. One type is the record player shown in Figure 3. The record sticking out of the cover on the right is a *Grabbable Object*, the player a *Socket Interactor*. Players can grab the record and place it on the player. If done correctly, two *Push Buttons* are enabled. Real record players usually have two speed settings, one for long players and one for singles. Depending on which button is pressed, the corresponding speed is simulated. Figure 4 shows the inspector of the button playing the faster track. Two events are used to play the track. The first *Audio Event* has the action *Set Audio Clip* selected, which tells the *Audio Player* to use the sped-up version. The second event triggers the *Audio Player* to start playing. The group also implemented a Walkman and a cassette player analogously. As this was the first time a larger group of developers used the tool, some small bugs occurred during the use (e.g. *Socket Interactors* snapping to the wrong position), which we will not describe in detail here.

Figure 3: Record Player

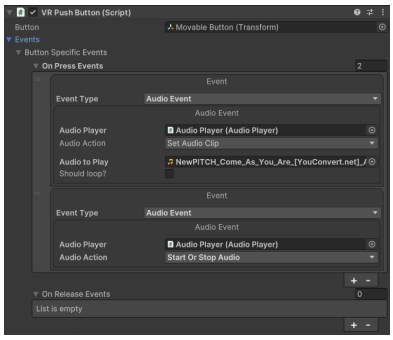

Figure 4: Event Config

Learnings We introduced the students to the tool at the start of the semester, before they started the conception phase. We wanted them to know the possibilities while creating the concept. But, some time passed between the introduction and development. Additionally, when starting development, students first created the models and levels, and even more time passed before they started creating interactions or logic. Hence, we needed to redo an introduction to the creation of interactions and logic. For some logic which is more complex then simple if-then relations, some additional explanation was necessary. However, after a short second introduction and some guidance in the beginning, the students were able to create the majority of their application on their own. This indicates the tool itself is rather easy to use, but we need to work on the initial explanation, so it is easier to get started with development. We provided a small

demo scene and textual description of the different objects, but students wished for a more sophisticated demo and manual. Also, we noticed that, in comparison to the previous semester where *Spatial* was used, the students managed to develop way more interactive applications while not needing more support. When the final version of the tool is released, we plan to perform a more in-depth evaluation, where we evaluate our tool against other available tools (like Spatial) and Unity's *XR Interaction Toolkit Examples* without the layer our tool provides.

5. Summary and Further Development

In this paper, we presented the concept of a course for our media study students, in which they create a VR application which can be used for other courses during their study. We defined our requirements based on guidelines for EUD and created a first version of a tool which fulfills them. We did an initial evaluation during the latest iteration of the course, which resulted in positive feedback, all students were able to create the applications they planned with little help. We found several software bugs which will be fixed during further development. Also, we noticed the need to prepare a better introduction, demo, and manual for the final version. Afterwards, we plan to perform a larger evaluation which measures its performance against other tools.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Y. Tan, W. Xu, S. Li, K. Chen, Augmented and virtual reality (ar/vr) for education and training in the aec industry: A systematic review of research and applications, Buildings 12 (2022). doi:10.3390/buildings12101529.

[2] H. Lieberman, F. Paternò, M. Klann, V. Wulf, End-User Development: An Emerging Paradigm, Springer Netherlands, Dordrecht, 2006, p. 1–8. doi:10.1007/1-4020-5386-X_1.

[3] H. Coelho, P. Monteiro, G. Gonçalves, M. Melo, M. Bessa, Authoring tools for virtual reality experiences: a systematic review, Multimedia Tools and Applications 81 (2022) 28037–28060.

[4] S. H. H. Shah, K. Han, J. W. Lee, Real-time application for generating multiple experiences from 360° panoramic video by tracking arbitrary objects and viewer's orientations, Applied Sciences 10 (2020) 2248. doi:10.3390/app10072248.

[5] Z. Zhao, X. Ma, Shadowplay2.5d: A 360-degree video authoring tool for immersive appreciation of classical chinese poetry, J. Comput. Cult. Herit. 13 (2020) 5:1–5:20. doi:10.1145/3352590.

[6] R. Blonna, M. S. Tan, V. Tan, A. P. Mora, R. Atienza, Vrex: A framework for immersive virtual reality experiences, in: 2018 IEEE Region Ten Symposium (Tensymp), 2018, p. 118–123. doi:10.1109/TENCONSpring.2018.8692018.

[7] S. Krois, E. Yigitbas, Prototyping cross-reality escape rooms, in: M. K. Lárusdóttir, B. Naqvi, R. Bernhaupt, C. Ardito, S. Sauer (Eds.), Human-Centered Software Engineering, Springer Nature Switzerland, Cham, 2024, p. 84–104. doi:10.1007/978-3-031-64576-1_5.

[8] E. Zidianakis, N. Partarakis, S. Ntoa, A. Dimopoulos, S. Kopidaki, A. Ntagianta, E. Ntafotis, A. Xhako, Z. Pervolarakis, E. Kontaki, I. Zidianaki, A. Michelakis, M. Foukarakis, C. Stephanidis, The invisible museum: A user-centric platform for creating virtual 3d exhibitions with vr support, Electronics 10 (2021) 363. doi:10.3390/electronics10030363.

[9] A. Bellucci, T. Zarraonandia, P. Díaz, I. Aedo, End-user prototyping of cross-reality environments, in: Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction, TEI '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 173–182. doi:10.1145/3024969.3024975.

[10] E. Yigitbas, J. Klauke, S. Gottschalk, G. Engels, Vreud - an end-user development tool to simplify the creation of interactive vr scenes, in: 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2021, p. 1–10. doi:10.1109/VL/HCC51201.2021.9576372.

[11] V. Frau, L. D. Spano, V. Artizzu, M. Nebeling, Xrspotlight: Example-based programming of xr interactions using a rule-based approach, Proc. ACM Hum.-Comput. Interact. 7 (2023). doi:10.1145/3593237.