Z3VR - An Interactive VR Learning Environment for One of the First Computers (Konrad Zuse's Z3)

Lukas Moersler, David A. Plecher(✉), Christian Eichhorn, and Gudrun Klinker

Chair for Computer Aided Medical Procedures and Augmented Reality,
The Technical University of Munich, Munich, Germany
{moerslel,plecher,klinker}@in.tum.de, christian.eichhorn@tum.de

Abstract. This paper presents Z3VR, an interactive simulation of Konrad Zuse's Z3 computer in a Virtual Reality (VR) environment, intended to give users the full experience of working with this historically significant machine. The paper explores other Z3 simulations and VR programming environments and draws comparison to Z3VR in terms of their accuracy and interface approach, where applicable. It details the creation of the project in various aspects and lessons learned from implementing and testing the VR user interface. It further evaluates how effective the project is as an educational tool in terms of teaching users about some concepts of low-level programming, and proposes future work to be done on Z3VR for higher usability and potential application in museums.

Keywords: Training education and tutoring · Serious XR · Serious VR · Zuse · Z3

1 Introduction

Virtual Reality (VR) is a technology that excels in creating immersive and engaging experiences, and which lends itself to letting users see and interact with places and objects they would otherwise not be able to. The Z3 computer, initially built by Konrad Zuse in 1941, is a prime candidate for such an environment. Clear historical significance aside, this machine is remarkable in both its accessible user interface and the numerous ways in which it mirrors modern processor and computer design. Among others, these characteristics make it a great example to teach users about the history of computing and low-level programming in a way that the skills learned can be translated into modern machine languages.

It should also be noted that, aside from other existing simulations (see *Related Work*), there is currently no possibility to work with the machine or even see it in person. The only full physical recreation at the *Deutsches Museum* in Munich is inaccessible at this time, as such this project aims to fill this gap by providing an immersive environment in which the Z3 can be seen from all angles and interacted with as if it were a real device. The main target demographic of the project is

J. Y. C. Chen and G. Fragomeni (Eds.): HCII 2024, LNCS 14708, pp. 123–141, 2024.
https://doi.org/10.1007/978-3-031-61047-9_8

people in their late teens and above, with at least marginal interest in computer science. Various interfaces are provided to make programming the machine as accessible as possible while trying to maintain immersion and historical accuracy.

The aim of this paper is to analyze the implemented VR interface in terms of its usability and to evaluate its effectiveness in teaching basic low-level programming concepts by analyzing data from play tests.

2 Related Work

Serious games have shown to be an effective approach to teaching in various fields [4]. By letting users participate in an immersive experience and giving them an active role in interacting with the subject matter, engagement and learning are amplified greatly. VR allows us to push the immersion aspect even further by fully enveloping users in interactive environments.

VR as a medium for serious applications and learning environments is a flourishing area of development [3,6], especially as the drop in cost of the necessary hardware in recent years has made it accessible for general audiences outside of research and military applications [3]. Serious games and applications using VR technology have successfully been used for teaching in a wide selection of disciplines, ranging from areas like public speaking [10,12] to physical assembly tasks [11], physics [26], virtual shopping [5,7,22] and even medical procedures [18]. And while VR is a cost-effective alternative to training in a physical environment, it also presents the unique opportunity to show places and objects which are normally inaccessible, be it because of insurmountable distances like different celestial bodies [1,2] or the content shown is in the past. Various prior works have dealt with the digital reconstruction of historical artefacts, such as underwater archaeological sites [13], reassembled ancient Greek statues [14], or the evolving state of historic buildings through the centuries [8]. In light of these examples, teaching programming, a traditionally text-heavy endeavour, may seem like an unusual application of this technology. However, Z3VR is far from the first project to experiment in this field [9,19–21].

The following will first cover the current extent of simulations of the Z3, followed by an examination of VR applications with similar goals to Z3VR in terms of historical simulations and teaching programming.

2.1 Previous Z3 Simulations

There have been various simulations and emulations of the Z3, with varying degrees of accuracy. How the Z3 operates has been covered in detail by Rojas et al. [17], for this section it's only necessary to know that it was programmed through punched tape, operated on two binary floating point registers, and it both received and displayed numbers through a console unit in a decimal floating point format.

2D and Terminal Simulations. A simulation by Mike Riley, published on their website[1], presents the user with an application window, interactive buttons and display fields mimicking the layout of the machine's console. Different tabs at the top of the window let users write programs, see the state of memory, and view debugging information. Riley's implementation executes faithfully the algorithms used to compute the various operations. The registers are abstracted as unsigned integers, and all the algorithmic steps as described by Rojas et al. [17] are taken to compute the results. The code for this was a valuable resource for checking the correctness of Z3VR's implementation of the arithmetic unit.

A different project named *pipZuseZ3*[2] runs entirely in a terminal and expects the name of a text file containing a Z3 program as an argument on startup. As such there is no option to simulate manual operations through the console. Since the program terminates after the instruction sequence has ended, the simulated machine's memory is not persistent, making programs that operate on the results of prior executions difficult. Furthermore, there is no option to create looping programs (possible on the real machine by sticking both ends of the punched tape program together), and the simulation does not implement the Z3's exception handling.

VRML Simulation. The prior work closest to Z3VR in terms of Z3 simulations is the 3D simulation of the machine available on the *Konrad Zuse Internet Archive* [15,16] (Fig. 1). It was constructed using the *Virtual Reality Modeling Language (VRML)*[3], used for displaying simple interactive 3D scenes embedded in web pages. Within it, users are presented with a scene featuring the Z3 machine and can navigate using the mouse via tank controls, or by switching between preset perspectives. The console can be interacted with by clicking on the buttons, and it mimics the real machine by displaying values and exceptions through illuminating the appropriate fields. Outside of the 3D view, the web page also features a section where users can write a program to be run, though this also does not offer the option of looping programs. Unfortunately, while the console interaction and programming both work as expected, the machine itself behaves in strange ways. Some of the buttons' functions seem to be swapped, and some don't appear to do anything at all. These issues are likely caused by incompatibilities with the software required to run this now unsupported format, and attempts at unpacking the VRML file to try and fix these were unsuccessful.

These three projects are currently the only publicly available simulations of the Z3. Since all of them are either non-functional due to unsupported formats or do not accurately represent the user interface, there is no possibility to accurately experience working with the Z3 (in an immersive way) - something Z3VR aims to change.

[1] http://www.historicsimulations.com/ZuseZ3.html, accessed 13.8.2023.
[2] https://sourceforge.net/projects/pipzusez3, accessed 13.8.2023.
[3] https://www.web3d.org/content/vrml97-functional-specification-and-vrml97-external-authoring-interface-eai, accessed 14.8.2023.

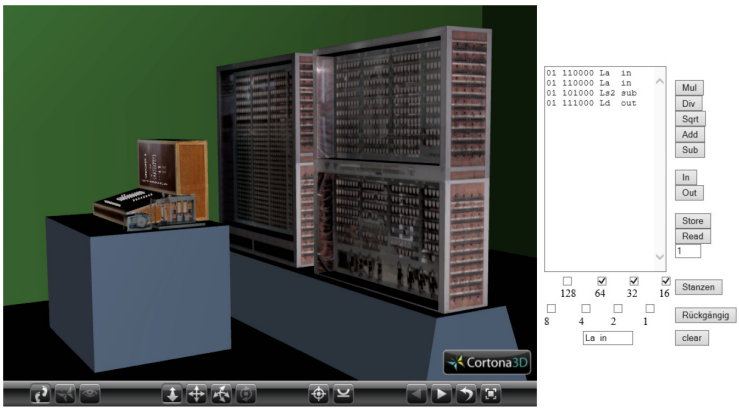


Fig. 1. VRML scene, navigable 3D scene to the left, programming view to the right

2.2 Other VR Simulations and Learning Environments

To put Z3VR into context, it's worth taking a brief look at other VR applications intended to simulate old computers or provide learning environments for programming.

ENIAC-VR. A remarkably similar project, only found after Z3VR's completion, is ENIAC-VR [25], presented at the MuC 2020 (Fig. 2). It had the same goal of providing an immersive virtual environment for users to experience programming one of the first computers, namely the ENIAC. This was extended further by including a guided tour which details the historical context and significance of the machine, as well as a *Maintenance mode*, in which users are instructed on how to spot and fix issues with the machine, which were a frequent occurrence on the real counterpart.

It features an introduction similar to Z3VR, which guides users by explicitly telling them what actions to perform to arrive at their first program, involving plugging in cables and turning knobs to a specific setting. Given the vastly higher number of interactable components, and far more complex programming model, ENIAC-VR makes heavier use of text-based instructions (Fig. 2), though it is difficult to imagine a purely visual or gesture-based tutorial similar to Z3VR's that could convey this level of information density effectively.

Block-Based Programming Environments. A notable commonality between the majority of VR experiences aimed at teaching programming is that the programming itself is usually done through the Block-Based approach popularised through services like *Scratch*[4]. This system has shown to be an effective tool for introducing students to programming while achieving a higher level of

[4] https://scratch.mit.edu/, accessed 14.1.2024.

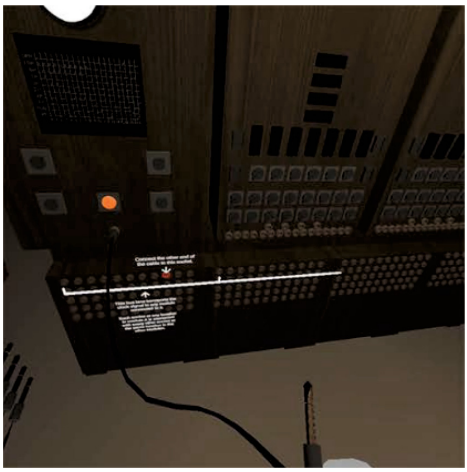


Fig. 2. Screenshot of the introduction to ENIAC-VR, text labels guide the user

engagement than traditional text-based programming [24]. A further common aspect is that the programs are often used to navigate a character through a level in much the same way as *Karel the Robot*[5], another established application for teaching basic programming.

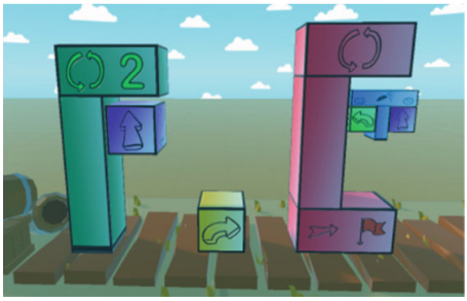


Fig. 3. A program in VR-OCKS, instructions are represented as cubes in a line

In *VR-OCKS* by Segura et al. [19] the code blocks which represent actions of the player character and concepts like loops are represented as literal blocks in the 3D scene which the user can pick up and place in a specific order (Fig. 3) to navigate their character around obstacles throughout a level. An earlier work by Vincur et al. [21], *Cubely*, presents effectively the same concept, but in the style of a popular video game franchise.

[5] https://xkarel.sourceforge.net/eng/, accessed 14.1.2024.

Z3VR took a similar approach to programming, as the programs are not written by the user in the traditional way, but are assembled from pre-set parts. However, an obvious difference is that Z3VR's pieces are merely textual entries in a 2D list, as will be detailed in the next section, and due to a lack of conditionals or finite loops in the Z3 instruction set, there are no "meta blocks" which contain a series of other pieces within them.

Fig. 4. A Program within Zenva Sky (Screenshots taken from https://www.youtube.com/watch?v=hMqiroApt2I, accessed 5.9.2023)

Zenva Sky also takes a similar approach to the prior two applications, however, in this case the user's programs control a vehicle in which they themselves are sitting [9]. Users have buttons in front of them that correspond to each possible instruction, and when they press them, they are added to an instruction list not unlike the one in Z3VR (Fig. 4). The game consists of a series of levels with increasing difficulty, in which the vehicle is used to reach an exit. Later the game is complicated by introducing locked doors and boolean inputs that the vehicle can activate, in combination with logic gates to open the doors.

In the first levels, the focus is on introducing the various logic gates; in later levels, the movement system is expanded with loops. The connection between the puzzles and their appearance in the code is shown by a screenshot of the corresponding Python[6] code in front of the user after each level.

A drastically different approach to teaching programming is provided by *Thinkercise*, a project by Theethum et al. [20]. The gameplay, seemingly inspired by *Beat Saber*[7], consists of hitting incoming coloured blocks with the correct hand, pictured in Fig. 5. Questions about Python (See Footnote 6) are regularly shown, with an upcoming set of three blocks being labelled with one possible answer each and the user being tasked with punching the correct one.

[6] https://www.python.org/, accessed 6.9.2023.
[7] https://www.beatsaber.com/, accessed 14.1.2024.

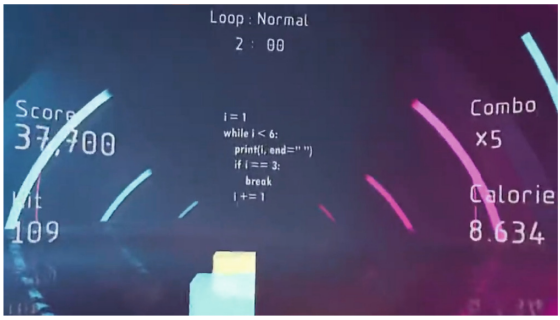


Fig. 5. Thinkercise gameplay, a programming question is presented while colored cubes approach the user (Screenshot from https://youtu.be/RFNkRtcWwas, accessed 14.1.2024)

This application is designed for users with at least rudimentary understanding of Python code, because the questions focus on advanced concepts of programming. Additionally, *Thinkercise* prompts the users to enter their weight to determine approximately how many calories they have burned during their session, adding a secondary motivation for users in addition to the educational aspect.

This project is far more gamified than Z3VR, and the focus of the training is on deepening existing programming knowledge rather than acquiring new skills.

3 Recreating the Z3 in VR

The following provides a quick introduction to the Z3 itself, to give context to all the VR interfaces which had to be implemented for it.

3.1 Z3

The Z3 was a programmable calculating machine operating on binary floating point numbers, completed by the engineer Konrad Zuse in 1941 [17]. Its operational model is effectively *Reverse Polish Notation*[8]: a mathematical operation requires first giving the operands and then the operator. The machine had instructions for receiving a number input through the console, displaying an output, accessing memory, and all four basic mathematical operations plus taking the square root. Calculations could be done in a "manual mode" by inputting numbers and pressing operation buttons. Alternatively, a reading unit could automatically run through a program stored on 35mm film stock with holes punched into it to encode instructions. The memory could only be accessed through such a program.

[8] https://en.wikipedia.org/wiki/Reverse_Polish_notation, accessed 1.9.2023.

Interaction with the machine happens almost exclusively through the console unit, which features a keyboard for number inputs and manual operation, and a light matrix for displaying results and exceptions. Fortunately, Zuse specifically set out to design his machines to be as user-friendly as possible [27], and working with the console is a simple process. A short introduction in which users are shown which buttons to press is enough to teach them how everything works within a few minutes.

3.2 Implementation

The project was implemented within the Unity3D game engine[9], which provides an OpenXR[10] API allowing the application to be used with VR hardware. Behaviours of objects within the engine are primarily governed by C# scripts. In Z3VR, each of the machine's separate units and peripherals has one script of that kind attached to emulate its function, with references to others allowing for communication where necessary.

The arithmetic unit is simulated on an algorithmic level, with the sets of relays representing numbers being abstracted as numerical values. The algorithms are executed within C# coroutines, which allows simulating the machine's speed (or lack thereof) by timing out the coroutine for the duration of a real Z3 processor cycle after each simulated one.

Aside from negligible technical details, a potentially inaccurate aspect of the simulated Z3 is its behaviour in various edge cases, which is largely undocumented. Z3VR implements a *watchdog* which monitors the current state of the machine and the upcoming instruction, and throws a generic exception on the machine's console if an invalid program sequence is detected. This system has shown to be a valuable aid for new users, as it prevents them from accidentally putting the machine in an invalid state. Such a system only requires a check of how many registers are currently filled and what the next operation expects, and as such is feasible to have been implemented on the real machine, but confirming this would require an extensive deep dive into its electrical plans.

An extra safeguard that clears both registers if a new program is inserted was certainly not part of the Z3's design, but was also necessary to stop users from putting the machine in an invalid state.

3.3 Interface

Programming in VR is a unique challenge which has been previously tackled through block-based programming for the most part as shown in the related work. Z3VR took a similar approach, but with some alterations. The requirements for creating Z3 programs both simplify and restrict the possibilities for

[9] https://unity.com/, accessed 30.8.2023.
[10] https://www.khronos.org/openxr/, accessed 30.8.2023.

the programming interface. On the one hand, as the Z3 had no if-statements or jump instructions, there is no need for complex systems that allow nested or branching scopes. On the other hand, since these programs are not used to, for instance, guide a character to the end of a level, they must be able to get arbitrarily long, so blocks that have to be physically assembled by hand would quickly become very cumbersome. The implemented interface will be detailed in the following.

In the real world, the Z3 was mainly programmed on paper. "Calculating plans" (Rechenpläne) were written by hand, with tables to keep track of which variables and constants are in which memory cell, and annotations describing what number inputs were expected at which time. These plans were then translated to punched film, though notes about the program had to be included to tell the operator how to use it properly. Such notes were especially important if a calculation required the sequenced execution of several programs in a specific order.

Z3VR provides interfaces and interactive elements to emulate this process without forcing the user to write anything by hand, and to and to minimize typing. These systems are introduced step by step to not overwhelm the users. At the beginning, users are placed in a bare scene in which they receive a quick introduction to the movement system and all types of interactions needed.

Interactable objects in Z3VR can be categorized as such:

- **Push buttons** → Physical buttons that gradually retract when the user places their virtual finger on them. As soon as a threshold value is reached, they fully engage and trigger whatever they are linked to. Examples include the console keyboard and the lever buttons of the film puncher.
- **Grabbables** → Objects that can be picked up using the trigger button and carried around. They are unaffected by gravity and can be placed anywhere or put into specialised *deposits*, which can have various functions. Examples include sticky notes and program films.
- **Openables** → Grabbables that can be opened by tilting them such that their spine points downwards. These include an info/trivia book and program folders.
- **2D UI buttons** → Standard Unity UI buttons on world space canvases. A raycast visualised by a laser pointer attached to each hand acts as the user's cursor, with the trigger button acting as the click action. This laser is only visible when aiming at the canvas.

Other interactions produce a small haptic feedback in the respective controller, to further enhance the immersion and responsiveness of the interface. Of note is also that the user's controllers are represented within the application as hands attached to their body with arms using simple inverse kinematics. These hands provide a rudimentary digital twin, and further act as an indicator for the type of interaction. This is currently available when the user's pose is changed next to an interactable object. These poses include a pointing gesture with the index finger extended near push buttons, a further opening of the hand near

grabbing objects, and the manifestation of a laser pointer while aiming at a 2D canvas to create an appropriate origin for the displayed laser.

The most common items that users will be interacting with are the program films. They're grabbables that open a 2D UI display of the program held within when approaching them, shown in Fig. 6. This display also features two buttons with which users can toggle to a plain text view (eye icon) and whether the program should loop or not (∞). The plain text view translates the instruction code names given by Zuse to a literal name describing what it does, for instance, *Ls1* becomes *Add*.

The book, referred to as the *Info Book*, provides extra information and trivia about various things, including individual sections of the machine and background info about how and why film strips were used for programming. The text within changes based on which marked section of the machine the user points at while holding the book.

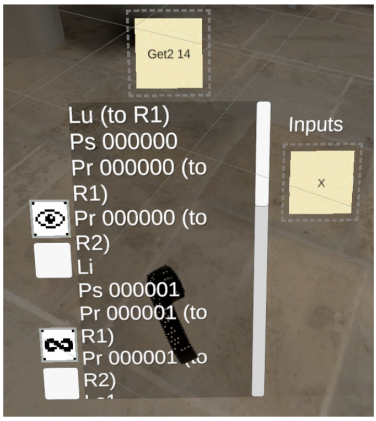


Fig. 6. Instruction list (middle) and sticky notes for name (top) and input (right)

Sticky notes were added as a simple and intuitive way to name programs and metaprograms for saving, and to define the expected inputs of a program. A large typewriter-style keyboard is provided to allow users to write on them. Even if it is not an optimal interface for writing in VR, it was sufficient, as the content of these notes is rarely more than one or two words. The program display that opens when holding a punched film has an outlined box at the top. If a sticky note is placed there, the program is saved under this name. The display also allows users to add comments to input instructions using sticky notes. These comments will appear within the *Info Book* if it is placed on the console while running a program.

The Z3 does not have an instruction for immediate values, so the only way to obtain constants is through user input. In addition, the machine has no way of

telling which variables should be entered in which order, which can be arbitrary depending on the program. The sticky notes were implemented to emulate the hand-written notes used back then while being as simple to grasp and use as possible. However, this system is not sufficient to communicate sequences of programs to the operator, so program folders were added.

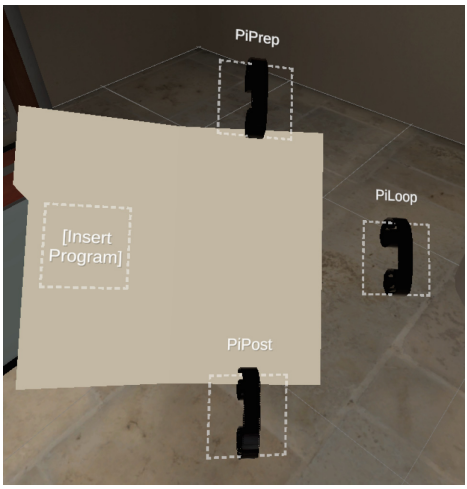


Fig. 7. Program folder containing three programs for approximating Pi

Program folders are used to store program films in a defined sequence. They can be opened the same way as the *Info Book* and offer a clock-like arrangement of boxes where programs can be placed, shown in Fig. 7. The number and positioning of these slots adjust automatically depending on the number of programs included. The folders themselves can be named and saved as well, to make the entries in the list of saved programs more compact.

Once the users has completed the interactive introduction, they are transported to the main scene, in which the Z3 is situated. At this point, the console does not feature any of the operation buttons, and the only option the users have is to start selecting a number by using the mantissa and exponent keys. Once a number has been selected, a small tutorial will start, guiding the user through their first calculation of $A + B$. There is no text telling them what to do, instead translucent hands indicate the action to be performed, in this case mainly pressing the operation buttons (Fig. 8). The next step does not start until the previous one has been performed. The only text added to the console is large numbers which reflect the currently selected number input and output number in the format $+1234 * 10^5$. This was added because some early testers had difficulty recognising the small button labels. These help texts can be disabled through the options menu. An additional aid in this respect is the option to double the size of the keyboard, which users are reminded of by an additional text on the console that disappears after the first number is entered.

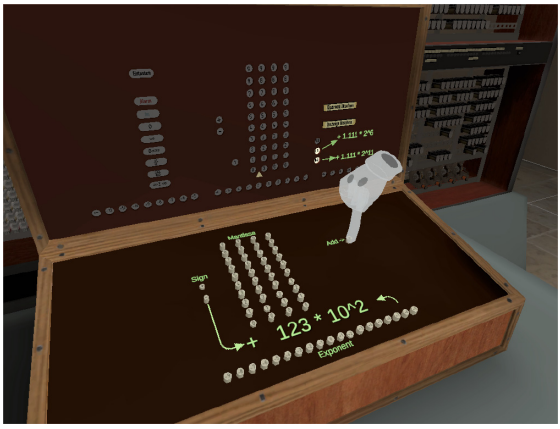


Fig. 8. Console introduction. A hand is shown telling the user to press the *Add* button.

This introduction system, internally titled *Step Tutorials*, has been implemented as a finite state machine, in which each state defines which scene objects should be enabled and disabled. Once the simple console introduction has been completed, all the remaining operation buttons appear, and the next scene element is shown to the user. This is indicated by a green arrow pointing to the next tutorial, which only appears when the objectives have been achieved. This indicator should be conspicuous but unobtrusive so as not to push the player to the next station, but to encourage them to experiment with what they have learnt previously.

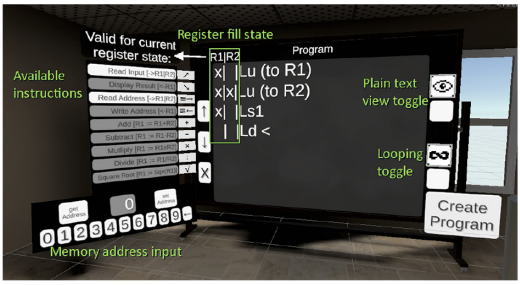
The programming board (Fig. 9a) is the main utility for writing programs, and implements the aforementioned simplified Block-Based programming. A scroll view in the middle shows the current list of instructions, and the left-hand side features buttons for each of the available operations. Clicking on one of these operations will add it to the instruction list where the cursor is located (indicated by <, can be moved by clicking on list entries). Further buttons allow for toggling a plain text view and program looping, as well as instantiating a punched film object with the current program.

As seen in Fig. 9a, some operations are greyed out. The *watchdog* is also used here to determine which operations are valid. The columns for $R1$ and $R2$ show the user which registers are filled after each instruction. If an invalid sequence is created, whether by inserting or deleting an instruction in the middle of the program, the invalid instruction will be marked red.

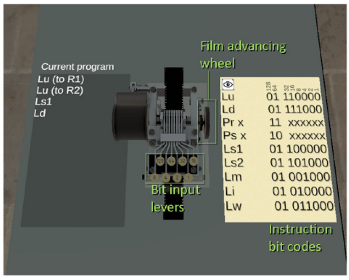
A film puncher is also present in the scene and an accurate reproduction of the device used for creating programs for the real machine. It can be used to punch films by hand in Z3VR as well, to get the full experience of programming the Z3. Users press the lever buttons corresponding to the bits of the current instruction code, then click the wheel on the right-hand side to advance to the

next instruction (Fig. 9b). A list to the left of the puncher shows what instructions have been punched, so users can see if they've made a mistake right away.

Further interaction stations are placed in the scene to allow the removing, saving or loading of programs.



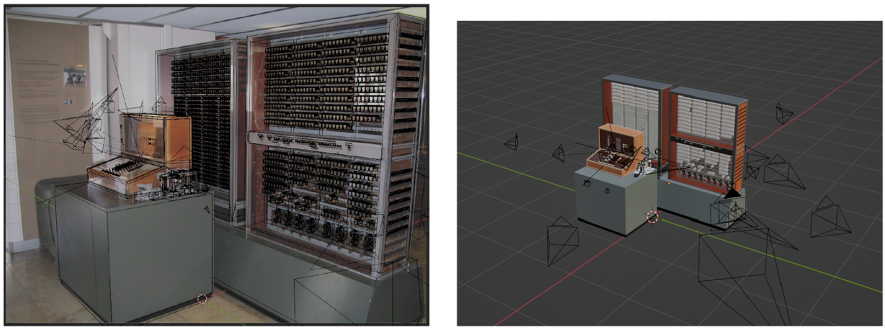
(a) Programming board, featuring the current program (middle) and available instructions (left). Invalid instructions are greyed out.



(b) Manual film puncher (middle) with instruction code reference (right) and currently punched program (left)

Fig. 9. Programming stations

3.4 Visuals



(a) View from one camera, 3D model visible as wireframe overlay, corresponding reference photo underneath

(b) Scene view of the 3D model surrounded by all reference cameras

Fig. 10. 3D recreation process

The Z3 model was recreated by extrapolating the position and orientation of cameras from online photos through *fSpy*[11], and importing these virtual cameras

[11] https://fspy.io/, accessed 21.8.2023.

into *Blender*[12]. Archived plans of the machine [28] revealed the height of the two relay cabinets, so an almost complete reconstruction of the machine was possible. (Fig. 10)

A notable detail of Z3VR is that the punched films have procedurally generated textures to reflect the program in them (visible in Fig. 6 behind the program display). Also an offset is applied to the texture for every step by moving the whole film forward by the length of one instruction.

3.5 Audio

Aside from greatly amplifying the immersion [23], audio provides feedback to the user. This ranges from the immediate response of a clicking noise when a button on the console is pressed, to the dozens of relays clacking away while the machine is crunching numbers. A video by the Deutsches Museum[13] was sampled to retrieve audio of the console buttons and the Z3 itself at work, with a short loop of the latter being played while the machine is calculating. To break up this loop and give at least some indication of which instruction is being run, an additional single-shot relay sound is played during memory access operations, implying the activation of the address decoder. When running a program, a loop of the pulse drum spinning is played while the machine waits for an input or is displaying a result, and is stopped when a program ends.

4 Evaluation

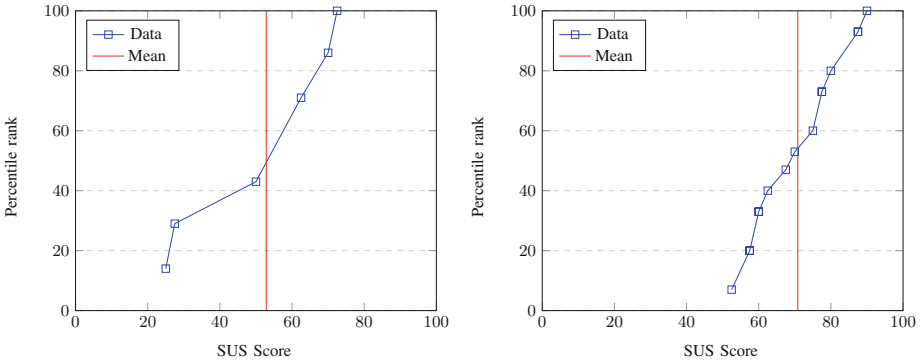
Evaluation of the project was done in the form of two testing phases. During these tests, each participant was guided through the application by the built-in introduction guide. They were then given a task in the form of a mathematical problem for which they had to create a Z3 program and were then given a questionnaire. This consisted of the *Standard Usability Scale (SUS)* questions supplemented with a few project-specific ones.

The first round of tests were run during development, just after a first draft of the introduction system had been implemented. At this point, it consisted of a series of text-bubbles, which the user could browse. All stations were present in the scene from the start. Some of testers simply skipped all the bubbles and were then left quite confused as to what to do.

To prevent this, the current version was introduced, which requires almost no text and introduces one thing at a time without distracting the user with unintroduced objects in the scene. The testing of this system was much smoother, the testers could follow the instructions and the necessary intervention of the test leader when the testers got stuck was much less frequent than in version 1.

Although the number of participants in each study was relatively small ($n = 7$, $SD = 19.548$ for version 1 and $n = 15$, $SD = 12.308$ for version 2), a clear

[12] https://www.blender.org/, accessed 21.8.2023.
[13] https://www.youtube.com/watch?v=aUXnhVrT4CI, accessed 21.8.2023.



(a) Version 1, $n = 7$ (b) Version 2, $n = 15$

Fig. 11. Percentile distribution of SUS scores across the two introduction versions

improvement in usability is evident looking at Fig. 11, with a shift of the average SUS score from 52.86 to 70.83. A two-sample t-test between the two data-sets confirms a significant difference: $t(20) = 2.64$, $p = 0.015$

During the testing of the second introduction iteration, 80% of participants chose to try and tackle the programming challenge. These were categorised into three levels of difficulty, with most subjects choosing the easiest. Nearly all subjects who took on a challenge completed it successfully, especially those who stated that they had no previous experience with low-level programming, suggesting that the introduction was successful in providing at least the minimum necessary to solve simple equations with hardware-level instructions.

5 Future Work

There is still room for improvement in some parts of the user interface, and with a few adjustments and additions, improving user-friendliness certainly seems possible.

Gamification. While some testers have already found the project entertaining, a separate game mode would be an interesting way to reward users for their learning and give them a sense of fulfilment. One suggestion that came up was a sort of “story mode” where the user is placed in an exaggerated scenario where an electromagnetic pulse has rendered all transistor-based technology inoperable and they must use the Z3 to calculate the trajectory of an incoming meteorite or something similar.

The gamification can of course be much simpler and more reasonable, like an interface in the scene supplying users with tasks (not unlike the challenges testers received) and rewarding users if they complete them, perhaps by unlocking further functionalities. Such a system could also be integrated into the introduction

system, giving users very simple tasks at first and then incrementally increasing the difficulty level to gradually imbue them with the mindset necessary to write programs for the machine.

General Improvements. An interesting addition would be a toggle-able abstract representation of the arithmetic unit that reflects the contents of the registers at all times, with the option to watch the algorithms at work in slow motion. Since the Z3's arithmetic unit is virtually identical to that of any current-day floating point processor [17], this would be a tremendous tool for teaching users about modern computers as well.

Only few people enjoyed working with the film puncher. Some testers had difficulty making the logical connection between the instructions, which are coded as bits, and the holes in the film, and of those who made the leap, only a few punched the program on the board in front of them. Obviously some work needs to be done on this interface to make it more accessible. One possibility would be to add a more rigorous version of the console's watchdog. A copy of the program on the programming board could be displayed next to the puncher, with commands that have not yet been punched greyed out and only the bit levers corresponding to the next command could be pressed at all. This would more clearly guide the user to press the correct levers and prevent them from accidentally punching an invalid instruction, which unfortunately often happens with virtual fingers that can easily slide over non-physical levers.

Address selection currently consists of a small text field below the instruction list with a numerical 2D UI keyboard, seen in Fig. 9a. While there are extra buttons for retrieving and setting the address of written memory instructions, this is still a very inconvenient system to use. A board containing sticky note deposits for each of the 32 possible memory cells could be a far better approach, being more accessible, more functional, better suited for VR specifically, and providing yet another use for sticky notes. Users can place notes to label addresses as you would on paper when planning a Z3 program by hand, to avoid keeping in mind which address corresponds to which variable or constant. The board also serves as a 2D UI, where clicking on a cell will transfer the corresponding address to a small text display next to the two memory instructions, and select it for adding either of those instructions to the program. While few testers tried programming with memory accesses, a further trial with this system would still be interesting, as the direct visualisation of the available memory might encourage its use.

Use in Musems. With some modifications Z3VR could be used as part of exhibitions in museums. A cooperation with the *Deutsches Museum* (where the real Z3 used to be on display) would be especially beneficial, as the real machine will not be demonstrated anymore due to fire safety concerns, and a full rewiring to fulfil requirements may not be possible[14].

To facilitate the usage in museums, a separate "guided tour" mode may be interesting. This would merely guide visitors around the different parts of the

[14] "Die letzte Rechnung der Zuse Z3" https://youtu.be/TbW-qNxD1lE.

machine with narrated descriptions and run some predetermined programs. The occasional genuinely interested visitor could still have the option to switch to the programming mode.

6 Conclusion

To return to the original research question: The current introduction system and programming interfaces have been shown to be effective, but still offer potential for further improvement. A follow-up study with more participants is needed to further investigate the capabilities of Z3VR for teaching low-level programming concepts. However, the test participants so far successfully applied what they learned in the introduction and in their own experiments when solving the math problems given to them, even those who stated that they had no previous experience with low-level programming.

Acknowledgments. This project would not have been possible without the plentiful resources provided by the *Konrad Zuse Internet Archive* [15], and especially professor Raúl Rojas, whose works on Konrad Zuse's calculating machines are a very good resource for anyone intending to learn about them.

References

1. Caravaca, G., Le Mouélic, S., Mangold, N., L'Haridon, J., Le Deit, L., Massé, M.: 3d digital outcrop model reconstruction of the Kimberley outcrop (gale crater, mars) and its integration into virtual reality for simulated geological analysis. Planet. Space Sci. **182**, 104808 (2020). https://doi.org/10.1016/j.pss.2019.104808
2. Casini, A.E., et al.: Analysis of a moon outpost for mars enabling technologies through a virtual reality environment. Acta Astronaut. **143**, 353–361 (2018). https://doi.org/10.1016/j.actaastro.2017.11.023
3. Checa, D., Bustillo, A.: A review of immersive virtual reality serious games to enhance learning and training. Multimedia Tools Appli. (2020)
4. Connolly, T.M., Boyle, E.A., MacArthur, E., Hainey, T., Boyle, J.M.: A systematic literature review of empirical evidence on computer games and serious games. Comput. Educ. **59**(2), 661–686 (2012). https://doi.org/10.1016/j.compedu.2012.03.004
5. Eichhorn, C., et al.: Inspiring healthy food choices in a virtual reality supermarket by adding a tangible dimension in the form of an augmented virtuality smartphone. In: 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), pp. 548–549. IEEE (2021)
6. Eichhorn, C., Plecher, D., Klinker, G.: VR enabling knowledge gain for the user (Venus). Technical report, TUM (2022)
7. Eichhorn, C., et al.: Shopping in between realities-using an augmented virtuality smartphone in a virtual supermarket. In: 2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp. 1161–1170. IEEE (2023)
8. Kersten, T.P., Tschirschwitz, F., Deggim, S.: Development of a virtual museum including a 4d presentation of building history in virtual reality. Int. Arch. Photogram. Remote Sens. Spatial Inf. Sci. **XLII-2/W3**, 361–367 (2017). https://doi.org/10.5194/isprs-archives-XLII-2-W3-361-2017, https://isprs-archives.copernicus.org/articles/XLII-2-W3/361/2017/

9. Navarro, P.F.: Zenva sky - VR app to learn computer science. Game published on Oculus store (https://devmesh.intel.com/projects/zenva-sky-the-world-s-first-vr-app-to-learn-computer-science) (2019)
10. Palmas, F., Cichor, J., Plecher, D.A., Klinker, G.: Acceptance and effectiveness of a virtual reality public speaking training. In: 2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp. 363–371. IEEE (2019)
11. Palmas, F., Labode, D., Plecher, D.A., Klinker, G.: Comparison of a gamified and non-gamified virtual reality training assembly task. In: 2019 11th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games), pp. 1–8. IEEE (2019)
12. Palmas, F., Reinelt, R., Cichor, J.E., Plecher, D.A., Klinker, G.: Virtual reality public speaking training: experimental evaluation of direct feedback technology acceptance. In: 2021 IEEE Virtual Reality and 3D User Interfaces (VR), pp. 463–472. IEEE (2021)
13. Plecher, D.A., Keil, L., Kost, G., Fiederling, M., Eichhorn, C., Klinker, G.: Exploring underwater archaeology findings with a diving simulator in virtual reality. Front. Virtual Reality **3**, 901335 (2022)
14. Plecher, D.A., Wandinger, M., Klinker, G.: Mixed reality for cultural heritage. In: 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), pp. 1618–1622. IEEE (2019)
15. Röder, J., Rojas, R., Nguyen, H.: Konrad Zuse internet archive. Website of the ZIB (2013). (http://zuse.zib.de. Accessed 14 Aug 2023
16. Röder, J., Rojas, R., Nguyen, H.: The Konrad Zuse internet archive project. In: Tatnall, A., Blyth, T., Johnson, R. (eds.) HC 2013. IAICT, vol. 416, pp. 89–95. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41650-7_8
17. Rojas, R.: Die Rechenmaschinen von Konrad Zuse. Springer, Heidelberg (1998). https://doi.org/10.1007/978-3-642-71944-8
18. Ruikar, D.D., Hegadi, R.S., Santosh, K.C.: A systematic review on orthopedic simulators for psycho-motor skill and surgical procedure training. J. Med. Syst. (2018)
19. Segura, R.J., del Pino, F.J., Ogáyar, C.J., Rueda, A.J.: VR-OCKS: a virtual reality game for learning the basic concepts of programming. Comput. Appl. Eng. Educ. **28**(1), 31–41 (2020)
20. Theethum, T., Arpornrat, A., Vittayakorn, S.: Thinkercise: an educational VR game for python programming. In: 2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), pp. 439–442. IEEE (2021)
21. Vincur, J., Konopka, M., Tvarozek, J., Hoang, M., Navrat, P.: Cubely: virtual reality block-based programming environment. In: Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology. VRST '17. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3139131.3141785
22. Walchshäusl, S., et al.: Generating an environment for socializing between older adults in a VR supermarket. In: INFORMATIK 2023 - Designing Futures: Zukünfte gestalten, pp. 325–337. Gesellschaft für Informatik e.V., Bonn (2023). https://doi.org/10.18420/inf2023_30
23. Warp, R., Zhu, M., Kiprijanovska, I., Wiesler, J., Stafford, S., Mavridou, I.: Validating the effects of immersion and spatial audio using novel continuous biometric sensor measures for virtual reality. In: 2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), pp. 262–265 (2022). https://doi.org/10.1109/ISMAR-Adjunct57072.2022.00058

24. Weintrop, D., Wilensky, U.: Comparing block-based and text-based programming in high school computer science classrooms. ACM Trans. Comput. Educ. **18**(1) (2017). https://doi.org/10.1145/3089799
25. Yigitbas, E., Tejedor, C.B., Engels, G.: Experiencing and programming the ENIAC in VR. In: Proceedings of Mensch Und Computer 2020. MuC '20, pp. 505–506. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3404983.3410419
26. Zengerle, T., Plecher, D.A., Flegr, S., Kuhn, J., Fischer, M.R.: Teaching optical principles in XR. In: INFORMATIK 2023 - Designing Futures: Zukünfte gestalten, pp. 313–323. Gesellschaft für Informatik e.V., Bonn (2023). https://doi.org/10.18420/inf2023_29
27. Zuse, H.: Die ergonomischen Erfindungen der Zuse-Maschinen im internationalen Kontext, pp. 95–120 (2008). https://doi.org/10.1515/9783839405642-002
28. Zuse, K.: Electrical plans of the rebuilt z3 (1960). Available on the Konrad Zuse Internet Archive http://zuse.zib.de. Accessed 14 Aug 2023