Virtual Reality Impacts on Novice Programmers' Self-efficacy

Nanlin Sun and Wallace S. Lages

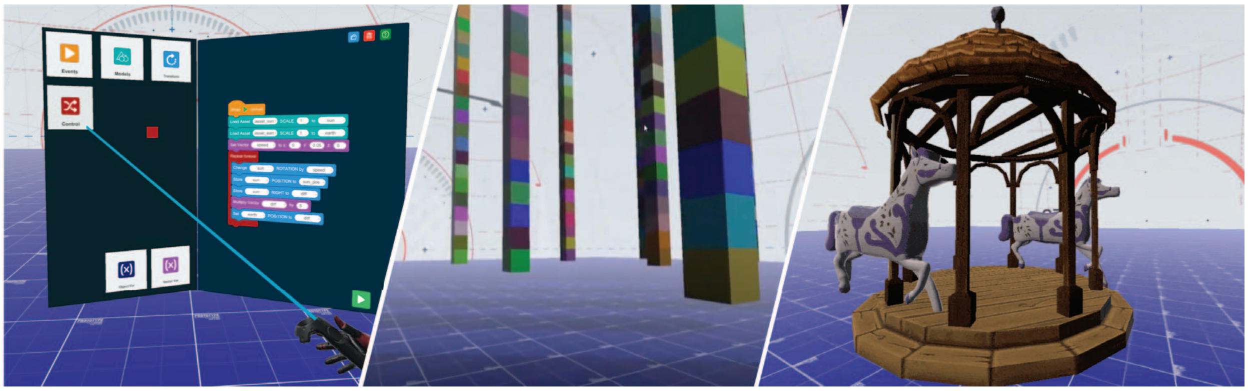


Fig. 1: Abacus immersive programming mode. Left: programming interface showing block selection and coding panel. Middle: example of procedural generation. Right: example of one of the study tasks, which requires animating existing 3D models.

Abstract—Virtual Reality has been used to improve motivation and help in the visualization of complex computing topics. However, few studies directly compared immersive and non-immersive environments. To address this limitation, we developed Abacus, a programming environment that can run in both immersive and non-immersive modes. We conducted a between-subjects study (n=40), with twenty participants assigned to the desktop mode and twenty participants assigned to the VR mode. Participants used a block-based editor to complete two programming tasks: a non-spatial procedural task, and a spatial 3D math task. We found that VR led to higher gains in self-efficacy and that the gain was significant for participants with lower initial levels of experience and spatial skills.

Index Terms—Virtual reality, programming, block-based languages, cs education

1 Introduction

Recent advances in display, processor, and sensor technology have renewed the interest in the educational use of virtual reality (VR) technologies. Virtual reality learning environments have the potential to improve student's learning in several areas. In particular, research indicates that immersive technologies may be effective in supporting spatial reasoning, a critical skill for many disciplines in science, technology, engineering, mathematics, and the arts [51].

Although most programming languages are spatially agnostic, many critical applications of computing happen in spatial domains [12]. In mechanical engineering, programming may be used to design and simulate 3D structures; in geographic information systems, to understand changes in cities and natural environments; in computer graphics, to animate characters for movies and TV [53]. With the expansion of augmented reality, 3D scanning, and 3D printing, spatial computation is also becoming increasingly integrated with the everyday physical space. Although spatial ability is correlated with programming skills [6], its interaction with learning in immersive environments has not been directly investigated.

Considering that virtual reality is an inherently spatial medium, we hypothesized it could significantly improve the learning experience of 3D programming. We define 3D programming as *the use of programming constructs to express algorithms that operate on 3D space*. In this paper, we investigate the impact of immersive environments on 3D programming. Our goal was to understand the impact of VR on the perception of competency and usability when compared to a similar desktop application. Specifically, we seek to answer the following questions:

- **RQ1: What is the impact of VR on student's perceptions of competency in programming?**

  Most prior studies of VR effectiveness for learning have focused on visual domains such as medicine or engineering. Although there are multiple results about the impact of 3D non-immersive programming tools, there is a very limited understanding of how these results would transfer to VR.

- **RQ2: How does VR impact the usability of block-based programming?**

  Text programming consists primarily of symbolic input, a task that is fairly inefficient in VR due to the lack of a keyboard. By using a block-based editor, we expected to reduce the input requirements enough to make the system usable.

To answer these questions, we developed Abacus, a tool that supports 3D programming in both VR and desktop modes. We used this tool to investigate how VR programming affects learners' beliefs in their ability to develop 3D programming code when compared to traditional desktop tools. Our results indicate that immersive programming virtual environments can support higher increases in self-efficacy than non-VR tools.

Our contributions are the following: (i) the design and implementation of Abacus, a visual 3D programming tool that allows both desktop and VR programming; (ii) a study examining how VR affects the self-efficacy of different learners' when accounting for prior programming experience and spatial ability; and (iii) the evaluation of the usability and perceptions of programming inside virtual environments.

- *Nanlin Sun is with Virginia Tech. E-mail: nannie@vt.edu.*
- *Wallace Lages is with Northeastern University. E-mail: w.lages@northeastern.edu.*

Received 18 September 2024; revised 13 January 2025; accepted 13 January 2025. Date of publication 10 March 2025; date of current version 31 March 2025. This article has supplementary downloadable material available at https://doi.org/10.1109/TVCG.2025.3549567, provided by the authors. Digital Object Identifier no. 10.1109/TVCG.2025.3549567

2 Background and Prior Work

2.1 Programming Self-Efficacy

Bandura introduced the concept of self-efficacy theory in an article published in the journal Psychological Review in 1977 [1]. Bandura defined self-efficacy beliefs (or expectancies) as beliefs about one's ability to do actions deemed required for achieving desired objectives. He suggested that self-efficacy beliefs are fundamental drivers of human behavior and presented the self-efficacy theory as a unifying explanation for multiple categories of behavior change, including the impacts of psychological therapies and psychotherapy. Standard instruments for measuring self-efficacy are the Motivated Strategies for Learning Questionnaire (MSLQ) [44] and the Computer Programming Self-Efficacy Scale (CPSES) [47].

According to Ramalingam et al., self-efficacy is more important as a determinant of introductory programming learning than other possible factors (intellectual ability, previous mathematics and computing experience, goal orientation, use of learning strategies, and nature of the conceptual model of the domain) [47]. Their study on computer programming self-efficacy found evidence that self-efficacy is more malleable during the learning process: novice programming learners are more willing to take on challenging tasks and display higher effort and persistence in achieving them. They also found that the student's self-efficacy was highly responsive to performance achievements in the early stages of skill attainment, particularly in students with initial low self-efficacy. Unfortunately, their study was designed primarily to target C++ programming learners and did not examine whether learning environments would also affect learners' perceptions of programming.

Virtual reality is a potential tool to increase learners' self-efficacy, since the emotional arousal created by high-fidelity simulations can change how one identifies with him/herself [20].

2.2 Immersive Learning Environments

Software tools play an essential role in helping students to learn. Learning environments can be non-stereoscopic (desktop applications), stereoscopic applications, or mixed. Most published studies focused on 3D desktop applications, which lack stereoscopy and head tracking, essential features for VR immersion [25, 30, 36, 37]. In a recent survey, only 13 papers published after 2013 evaluated VR systems [45]. For this reason, our understanding of the possible benefits of immersive programming is still lacking. Claims for explaining VR effectiveness often include: support for new forms and methods of visualization; unique first-person non-symbolic experience; increased motivation; and increased realism, which allows opportunities for insights from different perspectives [42]. To date, few studies explicitly modeled VR effects in comparison with non-VR environments.

Programming tools that use strategies such as games, storytelling, or different types of visualization aids are often preferred by the students [52]. There is some evidence that immersive learning environments can help novice programmers to better understand abstract computer science concepts, but there were no studies conducted to evaluate the potential benefits of actual programming or to obtain recommendations for optimal instructional design [4]. In a more general way, which type of programming would benefit the most from virtual spaces? In this study, we select the domain of 3D Programming, which we believe can benefit the most from VR's unique affordances. We also select one specific aspect of learning (self-efficacy) and derive a model that contrasts the impact of the VR environment when compared to a similar PC version.

2.3 Spatial Ability

The term *spatial ability* is used to describe multiple cognitive aspects related to capacity to understand, remember, or mentally transform visuo-spatial objects [60, 61]. High spatial ability has been associated with increased performance in visualization tasks in multiple domains, such as mathematics, engineering, and architecture. During the manipulation of objects, it can help to integrate and identify specific viewpoints [26].

Researchers have identified different factors, such as the ability to quickly rotate simple items, perspective change, or combine sequences of transformations [32]. One major factor separation is between small-scale spatial ability, which involves spatial manipulation at the object level, and large-scale spatial ability, which requires object manipulation at the environment level. Small-scale ability is measured by tasks that typically require an allocentric point of view (e.g., rotating an object in space) while environment-level requires an egocentric point of view (e.g., perspective taking) [27]. This difference seems to indicate that the person's ability to mentally manipulate a visual stimulus from a stationary point of view is distinct from the ability to reorient oneself in space.

Lages and Bowman found evidence that spatial ability affects the optimal way to interact with visualizations in virtual reality. When comparing walking vs hand manipulation, they found that participants with previous game experience but low spatial ability performed better using the manipulation technique instead of walking [29]. Barrera Machuca et al. followed a similar approach, this time to study the effect of the user's spatial ability on 3D immersive drawing [2]. They found that participants ranking higher in spatial ability achieved higher drawing performance, even when using a worse control method (hand-based). Recently, Drey et al. conducted a more detailed investigation of spatial ability impacts on VR manipulation [13]. They found evidence that higher spatial abilities resulted in significantly shorter task completion times and more targeted manipulations. They also found that lower spatial abilities can be compensated by improved interaction techniques.

In multimedia learning, multiple studies have shown the role of spatial ability in supporting learning. When comparing the performance of high and low spatial ability learners, some studies have found low spatial ability learners benefit most from 3D interactive learning environments [30, 56], but some found the opposite [31, 64]. Some have found that the effect is more significative for non-dynamic rather than dynamic visualizations, while others have found the opposite [19]. Carbonell-Carrera et al. studied the effectiveness of using a game engine for geospatial training [8]. The authors found that training to identify and locate landforms yielded significant improvement on some, but not all, perspective-taking measures.

In computer science education, the link between spatial ability and computing success was established with early evidence that learning BASIC could improve students' spatial skills [35]. Multiple subsequent studies found connections between different measures of spatial abilities (e.g., spatial visualization, perceptual speed) and measures of interest in computer science, such as program comprehension, indicating that they share an underlying ability [43].

To explain the link, Margulieux, proposed the Spatial Encoding Strategy theory (SpES), which posits that developing spatial skills allows people to develop better strategies for encoding non-verbal information and identifying landmarks for orientation [33].This in turn reflects in correlations with spatial ability and programming success.

In general, the interaction of spatial ability and visualization tasks needs to be further investigated, particularly in immersive learning environments. To the best of our knowledge, this is the first study connecting 3D programming and spatial abilities.

2.4 Tools for 3D Programming

Many of the concepts in 3D programming are difficult to understand and teach because they require connecting an abstract, mathematical idea (e.g., multiplying a vector by a quaternion) to an actual spatial operation (a rotation). In addition, reasoning about 3D operations (and programming in general) is correlated with spatial ability which varies in the population [50, 61].

One of the challenges of building programming environments for VR lies in the heavily symbolic nature of programming, which makes it hard to execute using traditional VR input devices. For this reason, effective programming environments often use visual languages for programming. Alice is a 3-dimensional interactive animation program visualization environment. Novice programmers build animated 3-D movies and author games while learning introductory object-oriented programming concepts. Alice has an extension under development that allows users to play worlds in VR, but it is limited to visualizing worlds built in the 2D tool [11].

Another example is the research programming environment 3D-VPL. It uses tridimensional blocks to represent classes and objects. Users can move around and add or remove objects to visualize programming concepts [39]. Unlike 3D-VPL, our proposal uses an actual block language and requires less user interaction on the manual code organization as the code lines grow larger.

My Reality (MYR) is another environment that combines programming with a virtual reality visualization [4]. However, MYR requires users to use a Web-Based platform outside VR to write code. Instead, we propose a learning environment where students don't need to switch back and forth between two platforms to modify their current work. We expect an all-in-one VR experience will lead to improved user experience and sustained embodied benefits.

2.5 Block-based Programming Environments

In the last two decades, block-based programming environments such as Scratch [49] and Alice [11] became part of the computer science educational landscape. Block-based editors allow users to program by manipulating visual elements instead of text. They have the advantage of removing typing issues and the need to memorize the language structure and syntax. Instead, users can browse the set of available commands from a palette and combine them. Command blocks have specific colors and shapes, providing visual cues on how they can be correctly combined [67]. In VR, the main advantage of block-based programming environments is the ease of composing scripts without using a physical keyboard.

Block-based programming has been adopted to lower the barrier to programming across a variety of domains, including mobile development [55], art [3], databases [62], robots [66], and parallel programming [16]. Research has shown that students become more interested in computer science after working in a block-based interface and that it can help with student retention in CS departments [38, 67].

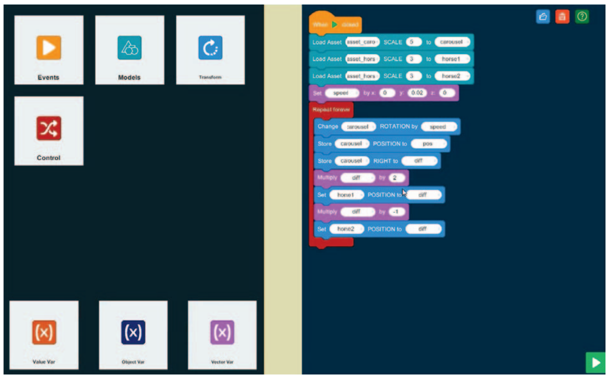
3 Abacus 3D Programming Environment

Abacus is a visual programming environment developed in Unity3D that allows the creation of procedural and interactive 3D scenes in both VR and desktop. Users can write scripts in block-based language to procedurally create interactive and dynamic scenes. We went through several iterations during the development process and evaluated different interaction techniques and back-ends. The final version implements user-adjustable floating UI panels which can be used to code, control execution, and see tutorials. The programming environment consists of: a) a block-based language interpreter, b) a set of 3D techniques for interaction, and c) UI interfaces for programming and auxiliary tasks.

In desktop mode, users navigate using a first-person controller. The W, A, S, and D keys are used to move, the space bar to jump, and the mouse controls the camera orientation. Interaction with the UI in the desktop mode uses a mouse and keyboard (Figure 2).

In the VR mode, users can move around using natural walking or teleportation. A ray attached to the controller is used to interact with UI elements, including a virtual keyboard with a standard QWERTY keyboard layout with 43 keys. Due to the use of blocks, the virtual keyboard is only needed to name variables. The floating UI panels allow users to manage different aspects of the coding process. The block panel displays the various blocks available for coding, grouped by categories. The variable panel enables users to create, delete, and inspect variables. The script panel is an empty canvas where the blocks can be freely placed. Multiple scripts to be placed on the same panel and executed simultaneously. It also allows users to save and load scripts (Figure 1, left).

We also created a curriculum viewer to support instruction activities, allowing users to hear audio instructions and animated task images. The tutorial audio clips can be played on the tutorial and task panel with AI-generated voice so that users don't have to read the lengthy text documentation. The audio player shows the current progress of the



```
When ▶ clicked
Load Asset asset_caro SCALE 5 to carousel
Load Asset asset_hors SCALE 3 to horse1
Load Asset asset_hors SCALE 3 to horse2
Set speed by x: 0 y: 0.02 z: 0
Repeat forever
    Change carousel ROTATION by speed
    Store carousel POSITION to pos
    Store carousel RIGHT to diff
    Multiply diff by 2
    Set horse1 POSITION to diff
    Multiply diff by -1
    Set horse2 POSITION to diff
```

Fig. 2: Abacus in Desktop mode. The screen is divided into two areas. Left: block menu. Right: script canvas. Once the play button is pressed (bottom right), the user can move around the 3D environment as in a first-person game.

tutorial clip, and users can mark a tutorial status as completed or incompleted to track the tutorial progress. Animated task images are also provided to visualize different checkpoints' expectations within instructional modules, which can help users progress step-by-step through a problem.

3.1 Blocks

Abacus executes visual blocks directly in native C# runtime without any middle layer interpretations, making the execution of Abacus reasonably fast. Abacus block editor was based on the Play Mode Blocks Engine [59], and each coding block is specified by a C# class. Coding blocks with similar instructions can share the same base C# class, making it easier to maintain and extend than plain text. Most coding block implementation bugs could be caught at compile time rather than at runtime, making troubleshooting easier.

Block-based editors, such as the one used in Scratch, use different block slot shapes to distinguish different values and variable types. Instead of requiring users to drag matching blocks into the block slots, some Abacus blocks use a selection menu that shows only the correct types. For example, a vector block requires variables of type vector. Instead of a block slot, a selection button in the block allows the user to quickly select the desirable variable from all the existing variables of this type. For the study, we created a special restricted mode (Task Mode), where only features relevant to our task were visible.

There are twelve block groups available in Abacus:

- The Event section contains blocks that respond to predefined events in Abacus. The event blocks are the entry points for their corresponding event. Thus, users can append code scripts at the end of them to be triggered when Abacus calls the event.
- The Models section contains blocks that help with using 3D models. Currently, Abacus supports two ways to build models: 1) loading existing models from the asset folder and 2) creating new objects from parametric primitive models.
- The Transform section contains blocks that manipulate the position, rotation, and direction of the object variable.
- The Control section contains blocks that have basic control over the execution flow of the coding script, including repeats (loops), breaks, waits, and if statements.
- The Interaction section contains blocks that can add interactions to the models stored in the object variable. This section was hidden in Task mode.
- The Visual section contains blocks that can change the visualization of the model stored in the object variable. This section was hidden in Task mode.
- The Sound section contains blocks to play audio clips (more advanced behavior-related blocks, such as audio control and spatial

audio, are scheduled as part of future work). This section was hidden in Task mode.

- The Operators section contains blocks for arithmetic operations and comparison operations. This section was hidden in Task mode.
- The Logic Gates section contains blocks that provide logic gates for Boolean operations. This section was hidden in Task mode.
- The Value Variables section contains buttons, windows, and blocks that help users define and manipulate value variables (variables containing a number or string). This section was hidden in Task mode.
- The Object Variable section contains buttons that help users define object variables (variables that contain a model).
- The Vector Variable section contains buttons, windows, and blocks that help users define and manipulate vector variables (variables containing a vector).

3.2 Preliminary Evaluation

To investigate the general usability of the system, coverage of blocks, and its suitability for creating creative content, we conducted two formative studies. The first was focused on evaluating the coverage of blocks and their suitability for creative work. We conducted a group exploratory session with 10 art students, sharing 4 VR workstations. Participants were initially briefed on the current state of VR for art creation, the motivation and contribution of the project. Then, they received information about the basic VR interaction and did a coding block tutorial inside Abacus. Next, the participants were asked to create something unique using any of the coding blocks provided. The session took approximately two hours.

The goal of the second study was to test the general usability of the VR and PC interfaces and evaluate the two tasks proposed for the main study. We invited three participants with limited programming experience to complete the tasks in both VR and PC modes. They were instructed to give feedback about the difficulty of the tasks and clarity of the instructions.

In the exploratory session, participants created different types of scenes by carefully composing existing objects in the scene and using loops to place a single asset multiple times. Collaboration between the participants emerged naturally, with participants taking turns in the VR headset and giving ideas and instructions by looking at the monitor.

Most participants in both studies showed a positive attitude towards using Abacus in the future. Students with prior block-based experience immediately recognized the Scratch-like interface in Abacus and were able to start programming immediately. Participants without previous experience in block-based programming spent more time understanding how each block worked and the need for the start special block. For the final study, we included two tutorials that explain how the blocks can be used together. We also renamed some blocks to make their function more clear. Finally, we improved the descriptions for the Programming Test of the Background Questionnaire, since it was unclear to one of the participants in the second study.

Although the early stage of Abacus did not provide as many coding blocks, we also observed that participants in both studies had difficulty locating blocks from different sections. For the final study, we implemented a focused interface that hides the blocks completely unrelated to the tasks. We believe this would not be an issue during actual, long-term usage.

4 Methods

To investigate our research questions, we conducted a study where participants completed coding tasks either in VR or PC. The study was approved by the Virginia Tech Institutional Review Board, protocol number 21-789. Written informed consent was obtained prior to the beginning of the study.

4.1 Participants

We recruited 46 university students from a large research university by sending recruitment information to several university mailing lists. Participation was restricted to individuals 18 years or older, able to follow instructions in English, with perfect or corrected vision (lenses or glasses), and basic programming knowledge. After expressing their initial interest in the study, participants received a follow-up email containing the link to schedule the 60-minute lab session, a link to the background survey, and a unique participant ID that would be required at the beginning of all surveys. Six participants did not complete the lab session and were dropped .

Of the 40 participants remaining, 33 were computer science majors, 2 were computer engineering majors, 1 applied statistics major, 2 mechanical engineering majors, 1 industrial systems engineering major and 1 from creative technologies. From the undergraduates, 1 was in the first year, 3 in their second year, 15 in their third year, and 11 in their last year. Of the graduates, 6 were in a master's program, and 4 in a PhD program. Of the participants, 9 had never used VR, 26 rarely used one, 4 used once a week, and 1 used weekly. Regarding the previous experience, participants had on average 4 years of programming experience and were able to understand simple code expressed as a block sequence (described in Section 4.4). Only two participants were not English native speakers. We did not collect gender or age information (most undergraduate students in the United States are between 18 to 22 years old). Table 1 lists the other relevant statistics.

Table 1: Minimum, mean, median, and maximum values for the number of years programming, programming test score, programming experience (questionnaire), and Cube Rotation Test score

	Min	Median	Mean	Max
Years Programming	1	4	4.59	13
Programming Exp.	47	100	93.76	100
Programming Test	0	100	83.82	100
Mental Rotation	-5	-10	9.82	19

4.2 Design and Procedure

We followed a between-subjects design, with 20 participants assigned to one of two conditions: VR or PC. In each one, participants independently completed two programming tasks: a procedural task (*Tree Forest*) and a math task (*Solar System*). The order of the tasks was alternated within each mode. Before each task, the experimenter guided the participants through related programming tutorials. Each participant completed a programming self-efficacy questionnaire three times throughout the study session: first, before the first tutorial (PreSE), the second, before the second tutorial, and the third, after the second task (FinalSE).

After, they completed the System Usability Scale (SUS), NASA Task Load Index (TLX), and User Experience Questionnaire Short (UEQ-S). A short three-question semi-structured interview was conducted at the end of the study. If the participant was assigned to the VR programming environment condition, an additional VR experience background questionnaire was completed at the beginning of the study (Figure 3).

4.3 Apparatus and Settings

In the VR condition, Abacus (VR Mode) was presented in the HTC VIVE Pro VR headset (1440 x 1600 resolution per eye, 98 degrees of horizontal field of view, 90 Hz refresh rate), with Vive standard controllers. Raycasting was used to manipulate the blocks, type in the virtual keyboard, and interact with interface controls. Participants were standing in a room-scale tracked area and allowed to physically walk or use teleport to move around. In the PC condition, Abacus (Desktop Mode) was presented using a standard monitor (1920x1080 resolution), mouse, and keyboard. Participants were seated and used a standard mouse and keyboard (Figure 2). Both conditions used, thus, the same software, blocks, menu layout, and programming interface.

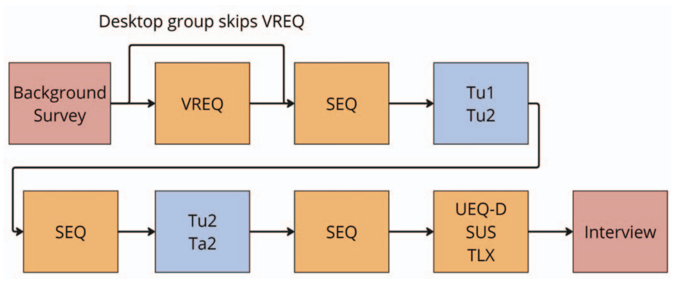


Fig. 3: Each participant must complete the background survey before arriving. Participants assigned to the VR programming environment group must complete the VR experience questionnaire (VREQ) upon arrival; the self-efficacy questionnaire (SEQ) is taken three times, with the first time before tutorial one (Tu1), the second time after task one (Ta1) and before tutorial two (Tu2), and the last time after task two (Ta2); the System Usability Scale (SUS), NASA Task Load Index (TLX), and User Experience Questionnaire Short (UEQ-S) are taken right before the interview at the end.

4.4 Background Survey

The study details and content page were listed on the first page of the background survey, which participants had to agree to before continuing. The background survey had six sections that took between fifteen to twenty minutes to complete and included demographic information, device usage questionnaire, self-efficacy questionnaire, the Cube comparison test [14], programming experience questions, and a block-based programming test.

4.4.1 Programming Concepts

Since 3D programming is a specialized programming domain, we included instruments to verify that participants were sufficiently comfortable with fundamental programming concepts.

The programming experience section was a modified version of the Measuring Programming Experience Questionnaire [15] where we included an additional table of programming concepts and asked participants to fill in the familiarity ("I am not familiar with it", "I've heard of it", "I know what it is", "I used it before" and "I am familiar with it") with each programming concept ("integer", "float", "boolean", "array/list", "variables", "assignment", "if statement", "switch statement", "for loop", "while loop" and "functions/methods").

We also included a programming tests section, with two simple programming quizzes written in a Scratch-like form One test was designed to check participant's understanding of variables. The second was designed to test the participant's understanding of the loops (the same set of movements has been repeated three times with the repeat block). Explanations of each block used in the two programming tests were also provided to the participants. Fig. 4 shows the Loop test.

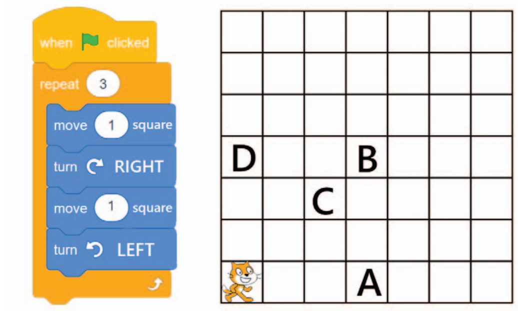


Fig. 4: The Loop test, one of the two programming tests in the Background Survey. We ask for the final position of the cat after the execution of the code snapshot on the left. A similar task was used check the understanding of variables.

4.4.2 3D Programming Self-Efficacy Scale

The Self-efficacy Scale consisted of 17 items. Following Bishop et al.'s Secure Programming Self-efficacy scale [5], we included two sub-scales: one for general programming and one to make it specific to 3D programming. For general programming, we used Bishop et al.'s sub-scale, removing one item that explicitly mentions computer science classes. For generality, we also included five items from Ramalingam and Wiedenbeck's "Independence and Persistence" sub-scale, so it could probe the individual's confidence to complete the tasks in different scenarios [48].

For the 3D programming sub-scale, we constructed five questions around meaningful fundamental tasks, such positioning and rotating objects:

- I can write a script to create one object in the 3D environment;
- I can write a script to create multiple objects in a line in the 3D environment;
- I can write a script to create multiple objects in multiple lines in the 3D environment;
- I can write a script to rotate and object procedurally;
- I can write a script to rotate an object around another object procedurally;

As in Ramalingam and Wiedenbeck, all items were scored in a 7-point Likert-type scale ranging from 1 to 7 (1 = "not confident at all," 2 = "mostly not confident," 3 = "slightly confident," 4 = "50/50," 5 = "fairly confident," 6 = "mostly confident," 7 = "absolutely confident"). The maximum score for the 3D Programming Self-efficacy scale is 119 (17 questions with 1-7 points for each question).

An analysis with pre-test data indicated good reliability, with a Cronbach Alpha coefficient of 0.825 for the whole self-efficacy scale consisting of 17 items; 0.838 for the 3D sub-scale consisting of 5 items; 0.807 for the general programming sub-scale; and 0.787 for the Independence and Persistence sub-scale consisting of 7 items.

4.4.3 Spatial Abilities Test

For measuring spatial ability, we selected the S2 - Cube Comparison test, from the Kit of Factor-Referenced Cognitive Tests from Educational Testing Service (ETS) [17]. In this test, assuming no cube can have two identical faces, the subject is asked to indicate which pairs of drawings can be of the same cube or not. It has two sections with 21 questions, each one administered with a 3-minute time limit. We selected a single test to keep the time commitment to the study manageable.

The Cube Comparison test was selected for being a standard and widely used test involving 3-dimensional figures. In addition, later work found that it is strongly related to one's ability to imagine the movement of an object or group of objects in an object-based frame of reference. This factor seems more pertinent to 3D programming than tests associated with perspective-taking and navigation. From the other three tests that loaded on this factor (Card Rotation (S-1), Paper Folding (VZ-2), and Guilford– Zimmerman Spatial Orientation (S-3)), the cube rotation had the highest load [27].

According to Carroll [9], the Cube Comparison test also involves factors related to the capacity of the visual short-term memory (STM). The performance seems to be mediated by spatial memory, so the results should be considered as also reflecting this factor [65].

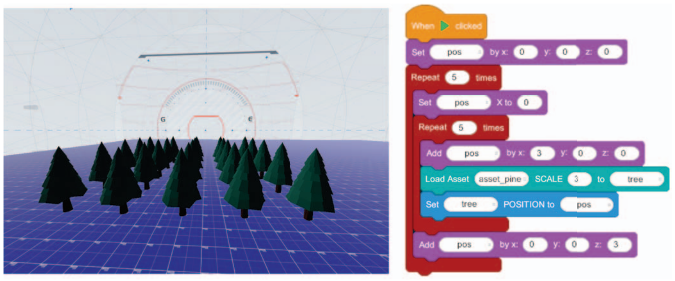
4.5 Programming Tasks

In this section, we explain the two 3D programming tasks used in the study. Both required participants to express spatial ideas in terms of code. Before each task, participants completed a tutorial that allowed them to explore the functionality of the blocks necessary to complete the tasks.

Procedural Task - *Tree Forest*

The procedural task focuses on the use of loops to compute equally spaced points in a plane. The goal of *Tree Forest* is to build a tree forest that has 25 trees and arrange them in the shape of a square of five by five. To accomplish this task without loading and placing each tree, participants need to combine finite loops with vector manipulations. The Pine tree is a model asset provided within Abacus and could be loaded with the Models-Load-Asset-Scale Block. The task was broken down into three steps to help participants proceed from start to finish. An image of the expected output is provided for each step for each participant to view at any time during the task.

For the first step of the procedural task *Tree Forest*, the goal is to create a single pine tree. A single Models-Load-Asset-Scale Block with a selection of the pine tree asset and the scale of one would be enough to complete this step. The reference code solution is not provided to participants during the task.



```
When ▶ clicked
Set pos by x: 0 y: 0 z: 0
Repeat 5 times
    Set pos X to 0
    Repeat 5 times
        Add pos by x: 3 y: 0 z: 0
        Load Asset asset_pine SCALE 3 to tree
        Set tree POSITION to pos
    Add pos by x: 0 y: 0 z: 3
```

Fig. 5: Procedural task *Carousel*. Left: The final goal of this task is to generate a forest by loading trees in the correct places. Right: an overview of a solution code.

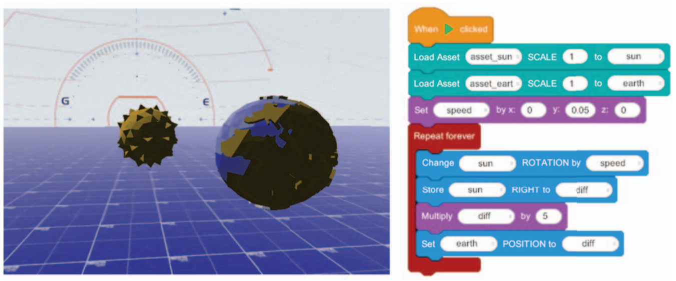
For the second step of the procedure task *Tree Forest*, the goal is to create one line of five pine trees. A finite loop of five is needed for this step, which could be done with the Control-Repeat-Number-Times block, as it performs the repeated process of creating a single pine tree five times. Additionally, a vector variable is needed to store and manipulate the positional data of each pine tree, which could be done within the VectorVar-NewVariable window. The VectorVar-Add-Variable block adds an offset to the positional vector for the spacing between pine trees. Finally, a Transform-Set-Position block is needed to set each pine tree to the positional vector each time after making a new pine tree. The reference code solution is not provided to participants during the task.

For the third and final step of the procedural task *Tree Forest*, the goal is to create a five-by-five-tree forest that repeats the previous step five times. Another finite loop of five is needed for this step which could be done with the Control-Repeat-Number-Times block, as it performs the repeated process of creating a line of five pine trees, which was accomplished in the previous step five times. Additional manipulations of the positional data are also required, adding an offset on the other axis of the vector for spacing on another dimension, which could be done with the VectorVar-Add-Variable block and resetting the previous dimension back to zero to create a new line of pine trees from the beginning, which could be done with the VectorVar-Set-Variable-X block or the VectorVar-Set-Variable-Z block. The reference code solution is not provided to participants during the task.

Before this task, participants were guided through a similar task: *Brick Wall*. In this tutorial, participants built a brick wall with 100 bricks arranged in the shape of a square of ten by ten. The tutorial has been broken down into three steps to help participants progress from start to finish: 1) create a single brick, 2) create a line of bricks with one loop, create a brick wall by nesting another loop. The reference code solution was provided to participants during the tutorial and could be accessed within the ToDo Panel.

Math Task - *Solar System*

The math task explores the use of vectors and rotation in spatial programming. The goal of *Solar System* is to build a solar system that has the Sun rotating around itself at the center and the Earth orbiting around the Sun. To accomplish this task, participants use infinite loops in tandem with directional vector manipulations. The Sun and Earth are provided model assets within Abacus and could be loaded with a Models-Load-Asset-Scale block. The task was broken down into two steps to help participants proceed from start to finish. A short video of what was expected at each step was provided for participants to view at any time during the task.



```
When ▶ clicked
Load Asset asset_sun SCALE 1 to sun
Load Asset asset_eart SCALE 1 to earth
Set speed by x: 0 y: 0.05 z: 0
Repeat forever
    Change sun ROTATION by speed
    Store sun RIGHT to diff
    Multiply diff by 5
    Set earth POSITION to diff
```

Fig. 6: Math task *Solar System*. Left: the final goal of this task is to generate an animated solar system. Right: an overview of a solution code.

For the first step of the math task *Solar System*, the goal is to create the Sun, which is an available model asset from Abacus, and then make it self-rotating. In addition to a Models-Load-Asset-Scale block to load the Sun, a vector containing the rotational data is required for the rotation, which could be done within the VectorVar-NewVariable window. Since the Sun is continuously rotating as the program starts, an infinite loop (Control-Repeat-Forever block is needed with the use of the Transform-Change-Rotation block to update the Sun rotation. The reference code solution is not provided to participants during the task.

For the second step of the math task *Solar System*, the goal is to make Earth move around the Sun. Earth only needs to be placed in the relative direction of the Sun, which could be obtained with the Transform-Store-Forward block or Transform-Store-Right block. To extend the radius of movement, a VectorVar-Multiple block is needed to multiply the magnitude of the direction vector. The reference code solution is not provided to participants during the task.

Before this task, participants were guided through a similar task: *Carousel*. The goal of this tutorial was to build a carousel with the center platform rotating around itself and two horses moving in the opposite direction around it. The implementation used an infinite loop and directional vector manipulations. The tutorial has been broken down into three steps to help participants progress from start to finish: 1) load the carousel model and make it rotate around itself, 2) create two horses and make them move in the opposite direction around the carousel. The reference code solution for each step was provided to participants during the tutorial and could be accessed within the ToDo Panel.

4.6 Interviews

At the end of each study session, we conducted a short interview with the following questions:

- Q1: Were the instructions for creating the forest and solar system tasks unclear? What additional information or clarification would you suggest to reduce confusion?
- Q2: Did the tutorial provided before the tasks (creating the forest before the wall, creating the carousel before the solar system) assist you in completing the tasks (creating the wall, making the solar system)?
- Q3: Is there any other feedback you would like to provide to help us improve the tutorial, tasks, or interface? Additionally, what potential future developments do you envision for this 3D programming tool?

5 Results

All analyses were performed using R Statistical Software (v4.3.2; R Core Team 2023) [46].

5.1 Descriptive Analysis

5.1.1 Self-Efficacy and Competency

We first report the correlation between programming competency, mental rotation, and self-efficacy for the sample group. We used the Robcor [54] package to calculate robust estimates of the coefficients. The correlation between mental rotation and years of programming was weak (.082). We also found a weak correlation between mental rotation and experience (.120). The correlation between the two competency measures was moderate (.500). The correlation between pre-test self-efficacy and experience was strong (.885), as was the correlation between pre-test self-efficacy and years of programming (.826).

After the test, self-efficacy increased on average in both groups, with the largest increase happening after the first task (Figure 7). The overall self-efficacy distribution was negatively skewed, so we used the Wilcoxon tests with continuity correction for a preliminary analysis. To compare the increase in self-efficacy, we used the Wilcoxon Rank Sum Test. The majority of the participants reported increased self-efficacy (V=561, p <.001). There was an overall significant difference after the first task (V=561, p <.001) and between the first task and the second (V=9, p <.001). To compare the self-efficacy between VR and PC we used the Wilcoxon Rank Sum Test with continuity correction. There was no significant difference between the rate of improvement of self-efficacy between PC and VR users (W=348, p=0.893). Table 2 lists the relevant descriptive statistics.

Table 2: Minimum, mean, median, and maximum values for Pre-test self-efficacy(Pre), self-efficacy after the first task (Mid), and self-efficacy after the second task (Final SE)

	Min	Median	Mean	Max
PreSE	34	78.0	79.29	119
MidSE	74	96.5	98.97	119
FinalSE	80	107.0	104.35	119

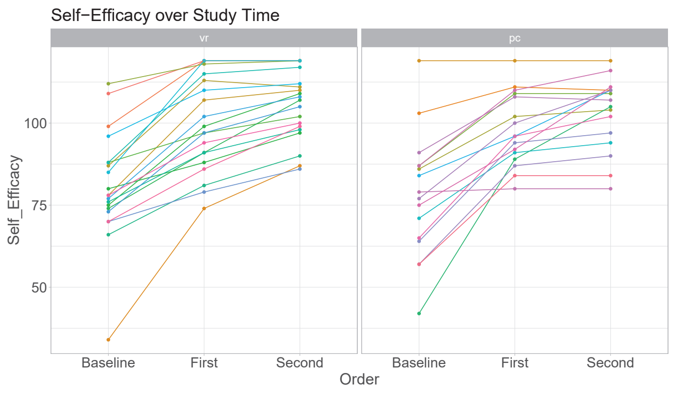


Fig. 7: Raw values of self-efficacy before and after each task.

5.1.2 Usability and Task Time

Regarding time, a Wilcoxon Signed Rank Test showed no significant differences between the time spent on the first and second tasks (V=342.5, p=.447). A Wilcoxon Signed Sum test also reported no significant difference between PC and VR for Task1 time (W=107, p=.255), Task2 time (W=120, p=.495), and total time (W=87.5 p=.069).

A Kruskal-Wallis Rank Sum test showed no significant difference between the participants with high and low spatial ability (mean 14 and 5 respectively), when using the mean Mental Rotation as a cut-point. This was true for Task1 time ($\chi^2 = 1.90, p = .59$), Task2 time ($\chi^2 = 4.20, p = .24$), and Total time ($\chi^2 = 6.00, p = .11$).

The median values (76.8) and mean (76.18) for the usability questionnaire of the system and the user experience questionnaire (3.07, 3.19) indicate above-average usability. A Wilcoxon Rank Sum Test showed no significant differences between VR and PC for SUS (W=162,p=.450), UEQ (W=151.5, p=0.700), and TLX (W=133, p=0.820). Table 3 shows the descriptive statistics.

Table 3: Minimum, mean, median, and maximum values for time in Tasks, System Usability Questionnaire (SUS), User Experience Questionnaire (UEQ), and NASA Task Load Index (TLX)

	Min	Median	Mean	Max
Time Task1	101	388.5	431.1	1244.0
Time Task2	116	333	382.0	104.35
Total Time	291	772	813.1	1537.0
SUS	57.7	77.5	76.18	100
UEQ	0.15	3.07	3.19	5.07
TLX	57.50	77.5	76.18	100

5.2 Model Analysis

Based on the prior work and our preliminary study, we hypothesized that prior programming experience and spatial ability would explain the effectiveness of VR in improving self-efficacy. Both contribute to reducing cognitive load and, consequently, to a better use of the VR features.

To test this hypothesis, we modeled the percentage increase in self-efficacy between the first and last measurements using spatial ability and prior block-based programming competency as co-variates of application mode (VR or PC). Using a ratio-dependent variable allows for easier comparative analysis, independent of the scale and absolute values. The test score was encoded as an interval variable. Mental Rotation and Experience were scaled and centered to ease interpretation. We used the glm function from the R Stats package for fitting and the DHARMa [18] package (v0.4.6) for model diagnostics. For model fitting, three participants were dropped because they were missing predictor data.

We compared three linear models with alternative proxy variables for competency: self-reported years of programming (Years), programming experience (Experience), and results of pre-test scores (PreTest). The full model using Experience had a significantly better fit (p=.0144 and lower AIC). Switching to a log link also improved the fit (p<.001 and lower AIC), possibly due to a nonlinear relationship between predictors and response. The final model selected was: $FinalSE/PreSE \sim MentalRotation * Experience * Environment$ (AIC=-21.306). Table 4 lists the coefficients of each predictor term.

We assessed the fit of the model regarding the collinearity of predictors, dispersion, zero inflation, and uniformity of the residuals, and outliers using the DHARMa library. Correlation between all the terms of the additive model were low (VIF $<$ 1.58), nonparametric dispersion tests of residuals distribution were non-significant (SD of residuals fitted vs simulated, dispersion=0.797, p=.4242), outliers (Exact binomial test, outliers=0, p=1), Dharma zero-inflation test (p=1), Uniformity (one-sample Komogorov-Smirnov two-sided test, D=0.13547, p=.5606). McFadden's R-Squared was 0.826. Together, the model diagnostic points to a good fit.

The fitted model indicates important contributions of the interaction between MentalRotation and Environment. When controlling for experience, individuals with lower spatial skills show higher relative improvement in self-efficacy in the VR than those in the PC environment. As spatial skills increase, the relative gain of both environments converges (Figure 8).

Regarding experience, the fitted model indicates that when controlling for spatial ability, gains in self-efficacy reduce as the prior experience of the individuals increases (or equivalently, that our intervention was more beneficial to those with less experience). The relative gains are higher for VR users, with the difference from PC users decreasing as experience increases (Figure 9).

Predictors	**Estimates**	***CI***	***p***
(Intercept)	0.34	0.27 – 0.40	**<.001**
MentalRotation	-0.00	-0.07 – 0.07	.966
Experience	-0.28	-0.40 – -0.16	**<.001**
Environment VR	0.08	0.02 – 0.15	**.008**
MentalRotation X Experience	0.13	-0.01 – 0.28	.089
MentalRotation X Environment VR	-0.08	-0.15 – -0.01	**.037**
Experience X Environment VR	-0.10	-0.23 – 0.01	.090
(MentalRotation X Experience) X Environment VR	0.11	-0.03 – 0.26	.143
Observations	34		
R^2	0.826		

Table 4: Estimates, confidence intervals, and p-values for the predictors of the model.

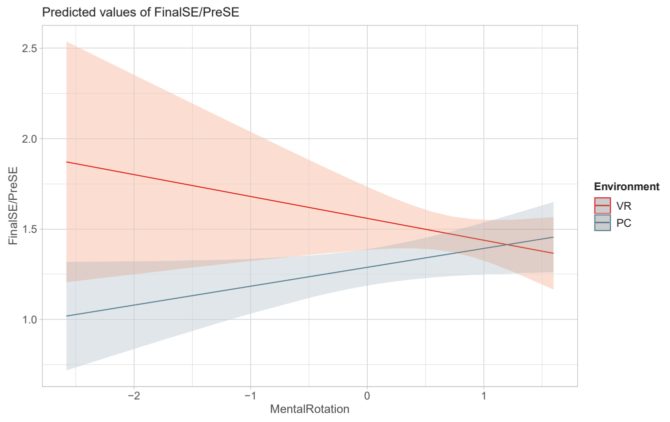


Fig. 8: Predicted values of the relative increase in self-efficacy (before and after the intervention). The model indicates that VR provides higher improvement rates than PC for individuals with low spatial skills.

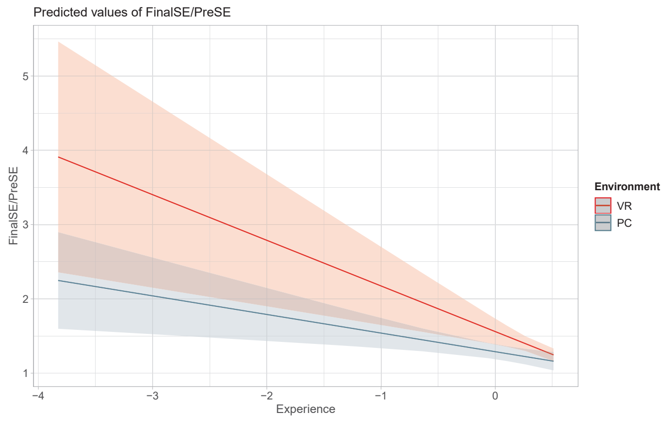


Fig. 9: The relative gains in self-efficacy reduce with experience. For individuals with low experience, gains from VR are higher.

5.3 Interview Results

Regarding the instructions used for the programming tasks, 20 participants found the instructions for the given tasks to be clear and not confusing; 6 participants suggested that the instructions could be improved to reduce confusion. Regarding the tutorial introduced before the programming tasks, 19 participants found the tutorial helpful in completing the tasks; 5 participants found the tutorial to be somewhat helpful. Several respondents suggested improvements to the 3D programming tool, including better code visualization and debugging tools, more helpful VR controls and interactions, and more interesting visualization options. In addition, many respondents suggested that the task instructions could be improved with more visual aids or examples to reduce confusion. There were also suggestions for more advanced features, such as physics simulations and real-time collaboration, and for more advanced tutorials for more complex projects.

6 Discussion

In this study, we directly compared the rate of improvement in self-efficacy in VR and PC. The findings indicate that VR was more effective than PC in raising participant's self-efficacy, particularly for those with lower spatial abilities or low experience.

6.1 RQ1: What is the impact of VR on student's perceptions of competency in programming?

Our fitted model indicates that self-efficacy gain in the VR condition increases as the participant's spatial abilities reduce. As discussed by Lee et al. and Mayer, the behavior of VR participants is congruent with the compensator hypothesis, which posits that participants with high spatial skills gain less from VR because they don't need the VR support to perform well [30, 34]. Kuhl recently proposed that the apparent inconsistency can be eliminated by assuming both as part of the same continuum where the difference between optimized and non-optimized designs are more pronounced for medium-ability learners [28]. At the higher end, high spatial skills dissolve the difference between VR and PC.

Our results are also similar to the study of Sun et al. where it was observed that low spatial ability learners had higher cognitive load (measured by amplitudes of N1 and N2 potentials) in the presentation slides-based environment than in the VR-based environment, implying that VR facilitates the reduction of cognitive loads in LSA learners. The P2 amplitude detected in HSA learners did not show any significant difference in both learning environments, indicating that the VR-based learning environment did not enhance their learning [56].

When looking at the effect of Experience, our model indicates that both VR and PC learners in the lower end of the distribution had higher gains. The improvement reduces at the higher end of the scale, where participants indicate familiarity with all the fundamental programming concepts. This could mean that students with more experience also had higher spatial ability, possibly developed through programming itself as discussed by Margulieux [33]. Instead, our model indicates a reduced gain when keeping mental rotation constant. This is further supported by the weak correlation found between spatial ability and experience/years of programming. However, the correlation between programming competency and initial self-efficacy was strong, which is in agreement with Bandura's self-efficacy theory (individuals with initial low efficacy, experience the greatest gains) [1].

Abacus makes use of block language primarily because of the ease of use in VR. However, by itself, block-based languages have a positive effect on learners' perceptions of ease of use of programming constructs by modality and interest in future computing courses as well [67].

According to the Cognitive Load theory [41, 57, 58] instruction can impose three types of loads on the learner's cognitive system: intrinsic load, caused by task complexity and prior knowledge; germane load, caused by features beneficial for learning; and extraneous load, caused by instructional features not directly linked to the learning [23, 40].

The overall cognitive load, thus, can be affected by both interface and instructional design. In our study, the intrinsic load due to the complexity of programming itself was left at minimal because we ensured participants had prior programming knowledge required to complete the task and scripts were short. The germane load consisted of the understanding of the 3D spatial operations required to accomplish the desired tasks, which was our goal. However, there was additional extraneous load: most of the participants were not very familiar with VR or block-based languages, the interface was new to all of them, and participants were able to freely explore the environment. These aspects may have impacted low spatial ability participants more.

6.2 RQ2 - How does VR impact the usability of block-based programming tasks?

Block-based programming environments such as Scratch rely on a block palette, from which the user drags the desired block. In addition, code refactoring requires selecting and repositioning specific blocks in the code. This kind of selection works well in desktops since the mouse is very precise. However, due to the lack of physical support, pointing accuracy in 3D space is reduced by motor tremors, and their effect is further amplified by distance [24, 63, 68]. Typing with a ray on a virtual keyboard is also more inefficient and frustrating than typing on a keyboard.

Because pointing and typing were replaced by raycasting, we expected to see a difference when compared to the same interface on mouse and keyboard. However, we were unable to find any significant difference in usability, as measured by the SUS, UEQ, and TLX. In fact, the scores for SUS and UEQ indicated an above-average usability. One of the reasons is that participants may have been able to prevent the controllers from degrading by avoiding interacting with the programming panel for a long distance. Our interface did not require a lot of typing, which may also have helped.

Another reason may be that participants' view of the system's usability may have been favorable. Merchant et. al [37] investigated the relationship between usability (latent variable with contributions from Perceived Meaningulness and Ease of Use), spatial orientation (as measured by the Purdue Visualization of Rotations Test [7]) and Self-Efficacy. Using path analysis, the authors tested the hypotheses that Usability influences spatial orientation and self-efficacy, finding significant positive relationships for both. In our view, their Usability construct confounds two different aspects (system features and psychological aspects). Although it makes sense to think that lower usability causes lower self-efficacy, participants may also find the system better if they have higher self-efficacy. It is also not clear how Usability might influence spatial orientation, which was only measured at the end of the study.

We also did not find a significant difference in task performance (as measured by time) between participants with high and low spatial ability scores. This contrasted with previous studies, which found that participants with higher spatial ability completed spatial tasks in a shorter time [2, 13, 29]. We believe this is because our tasks were relatively less demanding than the ones used in these prior studies.

It is interesting to note that several respondents found the instructions for the tasks to be straightforward and easy to follow, while others suggested that more specific or visual instructions would be helpful. This suggests that these learners may have different levels of spatial thinking skills and may benefit from different types of instructions. Additionally, the suggestion to improve the unit used in the programming environment and grid system to help users better understand the spatial relationships between objects in the virtual space is a valuable finding for improving the user experience. The suggestion to simplify the wording for programming blocks relating to vector and transform functions could also be valuable for programmers who may not be familiar with these concepts. Finally, the suggestion that the 3D programming tool could be used for both entertainment and education purposes is an interesting finding for future development and application of the tool.

7 Limitations and Future Work

Inferences based on few exposures, when participants are adjusting to the novelty of a medium can be misleading or incomplete. Since learners' motivation is partially determined by their perception of novelty, it may attenuate gradually over time due to familiarity [10, 21, 22]. Long-term studies are needed on novelty effects and the persistence of the improvements. This will help elucidate VR's sustained impacts on programming self-efficacy.

Due to the convenience sampling, we were not able to match or control the coverage of the covariates used in the modeling (spatial ability and experience). This power may be insufficient to investigate some of the moderation at all levels of these variables. In addition, because they are likely to affect the impact of VR learning, our findings may not generalize to other populations (e.g., younger learners, less experienced learners, or those with better spatial ability training). Future work should look into larger and more diverse samples and ways to improve the measurement and coverage of co-variates.

Future work should also look toward improving the instruments used. Our scale for 3D programming self-efficacy had good internal consistency (i.e. alpha = 0.70 or above). However, the scale can be further improved with further development, by revising or replacing some questions. In addition, our model used the S-2 (Cube Rotation Test) as a measure of (small-scale) spatial ability. It is possible that other factors may also contribute to 3D programming, which is left for future investigation.

Although in this work we focused on the impact of virtual environments on self-efficacy, VR may be more effective in enhancing the learning of 3D programming skills than non-VR media as well. Future work should look into domain knowledge acquisition in both procedural and spatial skills. In addition, a different set of tasks may help to provide a better coverage of 3D programming skills.

On the design side, the interaction techniques used in the Abacus block editor could be improved. Even though a ray is a straightforward translation of a mouse pointer that also conveys a simple metaphor, it becomes less effective when there is a need to select parameters for the blocks. Future work could look into better ways to edit block code, for example, by using direct manipulation or other interaction modalities; and exploring alternative ways to navigate the block library and compose scripts.

Finally, the present study used a block-based environment to allow for easier coding in VR and improve accessibility for learners. However, some of the benefits observed may translate to other visual and non-visual environments.

8 Conclusion

In this paper, we presented the design and implementation of Abacus, a visual programming environment that allows the creation of procedural and interactive 3D scenes in both VR and desktop environments. We investigated how VR impacts the learning experience of students learning 3D programming. Our findings indicate that, when controlling for spatial ability, participants with less experience showed higher gains in self-efficacy in the VR environment. When controlling for experience, individuals with lower spatial skills show higher relative improvement in self-efficacy in the VR than those in the equivalent PC mode. As the spatial skills increase, the relative gain of both environments converges. With the expansion of applications requiring 3D programming, immersive programming learning environments may help novices achieve higher levels of attainment.

Acknowledgments

This work was partially supported by a 4VA research award (# 12345-67890).

References

[1] A. Bandura. Self-efficacy: toward a unifying theory of behavioral change. *Psychological review*, 84(2):191, 1977. 2, 8

[2] M. D. Barrera Machuca, W. Stuerzlinger, and P. Asente. The effect of spatial ability on immersive 3d drawing. In *Proceedings of the 2019 Conference on Creativity and Cognition*, pp. 173–186, 2019. 2, 9

[3] D. Bau, D. A. Bau, M. Dawson, and C. S. Pickens. Pencil code: block code for a text world. In *Proceedings of the 14th International Conference on Interaction Design and Children*, pp. 445–448, 2015. 3

[4] C. Berns, G. Chin, J. Savitz, J. Kiesling, and F. Martin. Myr: A web-based platform for teaching coding using vr. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 77–83, 2019. 2, 3

[5] M. Bishop, I. Ngambeki, S. Mian, J. Dai, and P. Nico. Measuring self-efficacy in secure programming. In *IFIP World Conference on Information Security Education*, pp. 81–92. Springer, 2021. 5

[6] R. Bockmon, S. Cooper, J. Gratch, J. Zhang, and M. Dorodchi. Can students' spatial skills predict their programming abilities? In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 446–451, 2020. 1

[7] G. M. Bodner and R. B. Guay. The purdue visualization of rotations test. *The chemical educator*, 2(4):1–17, 1997. 9

[8] C. Carbonell-Carrera, P. Gunalp, J. L. Saorin, and S. Hess-Medler. Think spatially with game engine. *ISPRS International Journal of Geo-Information*, 9(03):159, 2020. 2

[9] J. B. Carroll. Psychometric tests as cognitive tasks: A new'structure of intellect'. Technical report, Princeton, New Jersey, USA, 1974. 5

[10] R. E. Clark. Reconsidering research on learning from media. *Review of educational research*, 53(4):445–459, 1983. 9

[11] S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, vol. 15, pp. 107–116, 2000. 3

[12] A. DeHon, J.-L. Giavitto, and F. Gruau. 06361 executive report–computing media languages for space-oriented computation. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007. 1

[13] T. Drey, M. Montag, A. Vogt, N. Rixen, T. Seufert, S. Zander, M. Rietzler, and E. Rukzio. Investigating the effects of individual spatial abilities on virtual reality object manipulation. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–24, 2023. 2, 9

[14] R. Ekstrom and U. O. of Naval Research. *Manual for Kit of Factor Referenced Cognitive Tests*. Educational Testing Service, 1996. 5

[15] J. Feigenspan, C. K"astner, J. Liebig, S. Apel, and S. Hanenberg. Measuring programming experience. In *2012 20th IEEE international conference on program comprehension (ICPC)*, pp. 73–82, 2012. 5

[16] A. Feng, M. Gardner, and W.-c. Feng. Parallel programming with pictures is a snap! *Journal of Parallel and Distributed Computing*, 105:150–162, 2017. 3

[17] J. W. French, R. B. Ekstrom, and L. A. Price. Manual for kit of reference tests for cognitive factors (revised 1963). *(No Title)*, 1963. 5

[18] F. Hartig. *DHARMa: Residual Diagnostics for Hierarchical (Multi-Level / Mixed) Regression Models*, 2022. R package version 0.4.6. 7

[19] T. N. Höffler. Spatial ability: Its influence on learning with visualizations—a meta-analytic review. *Educational psychology review*, 22(3):245–269, 2010. 2

[20] H. A. Holbrook and K. S. Cennamo. Effects of high-fidelity virtual training simulators on learners' self-efficacy. *International Journal of Gaming and Computer-Mediated Simulations (IJGCMS)*, 6(2):38–52, 2014. 2

[21] W. Huang. *Investigating the novelty effect in virtual reality on stem learning*. PhD thesis, Arizona State University, 2020. 9

[22] H. H. Ip and C. Li. Virtual reality-based learning environments: recent developments and ongoing challenges. In *International Conference on Hybrid Learning and Continuing Education*, pp. 3–14. Springer, 2015. 9

[23] M. Klepsch, F. Schmitz, and T. Seufert. Development and validation of two instruments measuring intrinsic, extraneous, and germane cognitive load. *Frontiers in psychology*, 8:1997, 2017. 8

[24] R. Kopper, D. A. Bowman, M. G. Silva, and R. P. McMahan. A human motor behavior model for distal pointing tasks. *International journal of human-computer studies*, 68(10):603–615, 2010. 9

[25] L. Korallo, N. Foreman, S. Boyd-Davis, M. Moar, and M. Coulson. Can multiple "spatial" virtual timelines convey the relatedness of chronological knowledge across parallel domains? *Computers & Education*, 58(2):856–862, 2012. 2

[26] M. Kozhevnikov and M. Hegarty. A dissociation between object manipulation spatial ability and spatial orientation ability. *Memory & cognition*, 29(5):745–56, Jul 2001. 2

[27] M. Kozhevnikov and M. Hegarty. A dissociation between object manipulation spatial ability and spatial orientation ability. *Memory & cognition*, 29(5):745–56, Jul 2001. 2, 5

[28] T. Kühl, B. C. Fehringer, and S. Münzer. Unifying the ability-as-compensator and ability-as-enhancer hypotheses. *Educational Psychology Review*, 34(2):1063–1095, 2022. 8

[29] W. S. Lages and D. A. Bowman. Move the object or move myself? walking vs. manipulation for the examination of 3d scientific data. *Frontiers in ICT*, 5:15, 2018. 2, 9

[30] E. A.-L. Lee and K. W. Wong. Learning with desktop virtual reality: Low spatial ability learners are more positively affected. *Computers & Education*, 79:49–58, 2014. 2, 8

[31] E. A.-L. Lee, K. W. Wong, and C. C. Fung. How does desktop virtual reality enhance learning outcomes? a structural equation modeling approach. *Computers & Education*, 55(4):1424–1442, 2010. 2

[32] D. F. Lohman. Spatial abilities as traits, processes, and knowledge. In *Advances in the psychology of human intelligence*, pp. 181–248. Psychology Press, 2014. 2

[33] L. E. Margulieux. Spatial encoding strategy theory: The relationship between spatial skill and stem achievement. *ACM Inroads*, 11(1):65–75, 2020. 2, 8

[34] R. E. Mayer. Multimedia learning. In *Psychology of learning and motivation*, vol. 41, pp. 85–139. Elsevier, 2002. 8

[35] R. E. Mayer, J. L. Dyck, and W. Vilberg. Learning to program and learning to think: what's the connection? *Communications of the ACM*, 29(7):605–610, 1986. 2

[36] Z. Merchant, E. Goetz, W. Keeney-Kennicutt, L. Cifuentes, O. Kwok, and T. Davis. Exploring 3-d virtual reality technology for spatial ability and chemistry achievement. *Journal of Computer Assisted Learning*, 2013. 2

[37] Z. Merchant, E. T. Goetz, W. Keeney-Kennicutt, O.-m. Kwok, L. Cifuentes, and T. J. Davis. The learner characteristics, features of desktop 3d virtual reality environments, and college chemistry instruction: A structural equation modeling analysis. *Computers & Education*, 59(2):551–568, 2012. 2, 9

[38] B. Moskal, D. Lurie, and S. Cooper. Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin*, 36(1):75–79, 2004. 3

[39] F. R. Ortega, S. Bolivar, J. Bernal, A. Galvan, K. Tarre, N. Rishe, and A. Barreto. Towards a 3d virtual programming language to increase the number of women in computer science education. In *2017 IEEE Virtual Reality Workshop on K-12 Embodied Learning through Virtual & Augmented Reality (KELVAR)*, pp. 1–6. IEEE, 2017. 3

[40] F. Paas and J. Sweller. Implications of cognitive load theory for multimedia learning. *The Cambridge handbook of multimedia learning*, 27:27–42, 2014. 8

[41] F. Paas and J. Van Merriënboer. Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of Educational Psychology*, 86(1):122, 1994. 8

[42] V. S. Pantelidis. Reasons to use virtual reality in education and training courses and a model to determine when to use virtual reality. *Themes in Science and Technology Education*, 2(1-2):59–70, 2010. 2

[43] J. Parkinson and Q. Cutts. Investigating the relationship between spatial skills and computer science. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pp. 106–114, 2018. 2

[44] P. R. Pintrich et al. A manual for the use of the motivated strategies for learning questionnaire (mslq). 1991. 2

[45] J. Pirker, A. Dengel, M. Holly, and S. Safikhani. Virtual reality in computer science education: A systematic review. In *26th ACM Symposium on Virtual Reality Software and Technology*, pp. 1–8, 2020. 2

[46] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2023. 7

[47] V. Ramalingam and S. Wiedenbeck. Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research*, 19(4):367–381, 1998. 2

[48] V. Ramalingam and S. Wiedenbeck. Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research*, 19(4):367–381, 1998. 5

[49] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. Silverman, et al. Scratch: Programming for all. *Commun. Acm*, 52(11):60–67, 2009. 3

[50] C. Roca-González, J. Martín Gutiérrez, M. García-Dominguez, and M. d. C. Mato Carrodeguas. Virtual technologies to develop visual-spatial ability in engineering students. *EURASIA Journal of Mathematics, Science and Technology Education*, 2017. 2

[51] P. Safadel and D. White. Effectiveness of computer-generated virtual

reality (vr) in learning and teaching environments with spatial frameworks. *Applied Sciences*, 10(16):5438, 2020. 1

[52] S. M. Salleh, Z. Shukur, and H. M. Judi. Analysis of research in programming teaching tools: An initial review. *Procedia-Social and Behavioral Sciences*, 103:127–135, 2013. 2

[53] S. Shekhar, S. K. Feiner, and W. G. Aref. Spatial computing. *Communications of the ACM*, 59(1):72–81, 2015. 1

[54] G. Shevlyakov and P. Smirnov. Robust estimation of the correlation coefficient: An attempt of survey. *Austrian Journal of Statistics*, 40(1&2):147–156, 2011. 7

[55] W. Slany. Tinkering with pocket code, a scratch-like programming app for your smartphone. *Proceedings of Constructionism*, 2014. 3

[56] R. Sun, Y. J. Wu, and Q. Cai. The effect of a virtual reality learning environment on learners' spatial ability. *Virtual Reality*, 23:385–398, 2019. 2, 8

[57] J. Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2):257–285, 1988. 8

[58] J. Sweller and P. Chandler. Evidence for cognitive load theory. *Cognition and instruction*, 8(4):351–362, 1991. 8

[59] Unity Technologies. Play mode blocks engine, 2019. Available at https://assetstore.unity.com/packages/templates/systems/play-mode-blocks-engine-158224. 3

[60] D. H. Uttal, N. G. Meadow, E. Tipton, L. L. Hand, A. R. Alden, C. Warren, and N. S. Newcombe. The malleability of spatial skills: a meta-analysis of training studies. *Psychological bulletin*, 139(2):352, 2013. 2

[61] M. C. Velez, D. Silver, and M. Tremaine. Understanding visualization through spatial ability differences. In *Visualization, 2005. VIS 05. IEEE*, pp. 511–518. IEEE, 2005. 2

[62] R. Vinayakumar, K. Soman, and P. Menon. Db-learn: Studying relational algebra concepts by snapping blocks. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–6. IEEE, 2018. 3

[63] D. Vogel and R. Balakrishnan. Distant freehand pointing and clicking on very large, high resolution displays. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pp. 33–42. ACM, 2005. 9

[64] D. Waller. Individual differences in spatial learning from computer-simulated environments. *Journal of experimental psychology. Applied*, 6(4):307–21, Dec 2000. 2

[65] L. Wang and M. Carr. Working memory and strategy use contribute to gender differences in spatial ability. *Educational Psychologist*, 49(4):261–282, 2014. 5

[66] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin. Blockly goes to work: Block-based programming for industrial robots. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, pp. 29–36. IEEE, 2017. 3

[67] D. Weintrop and U. Wilensky. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, 18(1):3, 2017. 3, 8

[68] J. Wither and T. Hollerer. Evaluating techniques for interaction at a distance. In *Wearable Computers, 2004. ISWC 2004. Eighth International Symposium on*, vol. 1, pp. 124–127. IEEE, 2004. 9