

Article

Design and Evaluation of Interactive Matrix-Based Algorithm Visualizations in Introductory Programming: A Student Perception Study

Ladislav Végh , Ondrej Takáč, Krisztina Czakoová and Dávid Demeter 

Department of Informatics, Faculty of Economics and Informatics, J. Selye University, 94501 Komárno, Slovakia; takaco@ujss.sk (O.T.); czakoovak@ujss.sk (K.C.); 127616@student.ujss.sk (D.D.)

* Correspondence: veghl@ujss.sk

Abstract

Algorithm development and its implementation by novice programmers in their first or second year of programming education can be difficult to teach because they often involve two-dimensional data structures, such as matrices. The use of these data structures requires both the ability to use nested iterations and the ability to coordinate row and column indices correctly. While algorithm visualization is commonly suggested as a method for externalizing intermediate program states and enhancing the understanding of algorithms, empirical studies have primarily focused on one-dimensional structures, such as sorting vectors using various algorithms. This study investigates the pedagogical potential of theory-informed, interactive, matrix-based algorithm visualizations implemented with the MatrixVis JavaScript library, a reusable web-based framework that provides step-by-step execution, visual highlighting of matrix operations, and synchronized source code display. The visualizations were demonstrated during introductory programming lectures and evaluated using a structured questionnaire ($N = 48$). Students evaluated the visualizations positively. The perceived usefulness of animations for understanding algorithms received a mean rating of $M = 4.56$, while the clarity of the visual presentation ($M = 4.58$), step-by-step execution ($M = 4.54$), and synchronized code highlighting ($M = 4.69$) were rated as particularly supportive. Visualizations of aggregation algorithms were perceived as the clearest ($M = 4.58$), whereas visualizations of structural transformations were slightly more difficult to follow ($M = 4.08$). These findings suggest that theoretically grounded, interactive visualizations of algorithms on matrices can provide meaningful pedagogical support in introductory programming education.

Keywords: algorithm visualization; programming education; interactive learning environments; matrix algorithms; educational visualization tools; student perception



Academic Editor: Nynke Bos,
Hanneke Theelen, Tjark Huizinga
and Suzan van Brussel

Received: 25 May 2026

Revised: 3 July 2026

Accepted: 14 July 2026

Published: 16 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Learning algorithms and gaining relevant algorithmic thinking skills during introductory programming courses present significant challenges for novice students. Understanding algorithmic processes requires learners to mentally simulate program execution, track variable updates, and follow the changing state of data structures across multiple computational steps (Robins et al., 2003). These difficulties become even more pronounced when algorithms operate on two-dimensional data structures such as matrices, where nested iteration, row and column indexing increase the cognitive demands placed on learners.

Algorithm visualization has long been proposed as a teaching method to support the understanding of algorithms by making the intermediate states of data structures and program execution observable (Brown & Sedgewick, 1984). It is commonly distinguished from program visualization, where algorithm visualization focuses on conceptual representations of algorithmic processes, while program visualization emphasizes the runtime behavior of specific program executions (Sorva, 2012). In educational environments, these systems aim to clarify abstract computational processes by making them more concrete, enabling learners to visualize and understand them. However, previous research suggests that the effectiveness of visualization depends not only on the animation itself but also on the level of learner engagement and the design of the visualization environment (Hundhausen et al., 2002; Naps et al., 2003).

Many pedagogical studies focusing on the use of algorithm visualizations in computer education have concentrated on one-dimensional data structures, particularly various sorting algorithms working on vectors (Shaffer et al., 2010). In comparison, notably less attention has been paid to visualizations of algorithms that use two-dimensional data structures, such as matrices, even though these structures are conceptually more complex and are frequently present in introductory programming courses. Therefore, this paper examines students' perceptions of interactive matrix-based algorithm visualizations developed with the MatrixVis JavaScript library and demonstrated in introductory programming lectures.

Based on this objective, the study addresses the following research questions:

RQ1. Overall pedagogical evaluation. How do students evaluate the clarity, usefulness, and overall pedagogical support of interactive matrix-based algorithm visualizations implemented with the MatrixVis JavaScript library?

RQ2. Design-feature-level support. To what extent do specific design features (step-by-step execution, visual highlighting, and synchronized code display) support students' perceived understanding of matrix algorithms?

RQ3. Differences across visualization categories. Do students report differences in perceived clarity and usefulness across different categories of matrix algorithm visualizations (aggregation, search, modification, and structural transformations)?

RQ4. Perceived value of interactivity and reusability. To what extent do students perceive interactive manipulation and the reusability of matrix-based visualizations as supporting their engagement and self-directed learning?

2. Literature Review and Theoretical Foundations

This section reviews the theoretical and empirical foundations relevant to the design and evaluation of matrix-based algorithm visualizations in introductory programming.

2.1. Algorithm Learning as a High-Cognitive-Load Domain

Learning algorithms is inherently cognitively demanding for novice learners. Students must mentally simulate dynamic state changes during program execution (Robins et al., 2003). Recent literature reviews on algorithm design education similarly highlight persistent difficulties in abstraction, state tracing, and multi-step reasoning, emphasizing the need for structured instructional support in early programming courses (Liu et al., 2024). Understanding algorithms involves following changes in variables, transitions in control flow, and the evolution of data states over time. Cognitive load increases when learners move from one-dimensional arrays to two-dimensional matrices because of the greater interactivity among elements (Sweller, 1988, 2011).

Matrix operations involve nested loops, two-dimensional indexing (the coordination of rows and columns), and global structural modifications, like row exchanges or transposition. These operations require learners to integrate symbolic code execution with spatial state

propagation simultaneously. From a Cognitive Load Theory perspective, such tasks are characterized by high intrinsic cognitive load due to multiple interacting elements (Sweller, 2011). Without an external representation of intermediate states, students must reconstruct each transformation step internally, increasing working memory demands. Visualization systems capable of externalizing the evolution of states may therefore reduce unnecessary mental simulation and support the understanding of complex matrix-based operations.

2.2. Algorithm Visualization: From Passive Animation to Active Engagement

Empirical findings indicate that visualization alone does not guarantee improved learning outcomes. Hundhausen et al. (2002), in a meta-analysis of 24 controlled studies, concluded that the effectiveness of algorithm visualization depends more on learners' active engagement with the visualization than on the animation itself. This insight was formalized in the engagement taxonomy proposed by Naps et al. (2003), which distinguishes among viewing, responding, changing, constructing, and presenting. The taxonomy was later extended by Myller et al. (2009) to account for collaborative engagement contexts. Grissom et al. (2003) empirically demonstrated an engagement gradient: students who actively responded to visualization prompts achieved significantly higher performance than those who were passive viewers. Similarly, Fouh et al. (2012) reported stronger learning outcomes at higher engagement levels, particularly in responding, changing, and constructing, while also noting that these levels require greater time investment and cognitive effort. Subsequent studies have similarly confirmed that passive viewing alone is generally insufficient to produce meaningful learning gains (Shaffer et al., 2010), while visualization combined with structured interaction can improve programming problem-solving performance (Prasad et al., 2021).

At the responding level, students who answered questions integrated into algorithm visualizations showed the most consistent improvement (Grissom et al., 2003). This pattern aligns with the generative activity principle of the Cognitive Theory of Multimedia Learning, which suggests that guided generative activities during learning foster deeper cognitive processing (Mayer, 2024). In this context, this type of interaction does not necessarily require an externally submitted or spoken response; it may also involve mentally formulating an answer, predicting the next step, or mentally evaluating the algorithmic state before the visualization proceeds (Hansen et al., 2002).

At the changing and constructing levels, learners manipulate inputs or build their own visualizations, which can be interpreted through the lens of Cognitive Load Theory: schema construction requires active cognitive processing and structured interaction with the learning material (Sweller et al., 1998). However, constructing and presenting activities require substantial intrinsic and germane cognitive load, which may exceed the available cognitive capacity of novice learners (Fouh et al., 2012). Thus, engagement is not linearly beneficial; its effectiveness depends on the balance between active processing and cognitive overload.

Several instructional approaches have attempted to implement these higher engagement levels in practice. Game-based and discovery-oriented approaches, for example, allow learners to experiment with the behavior of algorithms. Earlier research demonstrated that a game-based sorting environment enabled students to reduce redundant comparisons within a sorting algorithm and improve its efficiency through iterative engagement (Végh & Stofferová, 2016). Similarly, Yamashita et al. (2016) implemented a visualization system in which instructors defined visualization policies while learners could input data and observe the resulting algorithmic behavior. Such scaffolded discovery environments allow exploration while maintaining instructional guidance. This approach corresponds with Hmelo-Silver et al. (2007), who argued that inquiry-based learning is effective only when

appropriate instructional scaffolding is provided. Consistent with this view, recent reflections on algorithm education emphasize that experiential approaches, such as visualization and performance experimentation, can make abstract algorithmic processes cognitively concrete and pedagogically more accessible (Velázquez-Iturbide, 2025). From an interaction design perspective, effective learning environments must also make the effects of learners' actions clearly observable. Norman's (1988/2013) concepts of the Gulf of Execution and the Gulf of Evaluation highlight the importance of transparent system feedback and interpretable system states in interactive learning systems.

Consequently, the literature does not simply ask whether animation works but rather under what conditions of engagement, visualization supports learning. In this context, the MatrixVis framework supports selected forms of engagement, particularly viewing, stepwise exploration, and changing input values. The possibility of modifying input data and observing its immediate impact corresponds mainly to the changing level in the taxonomy proposed by Naps et al. (2003), while step-by-step execution supports guided exploration of algorithm behavior. The MatrixVis library used in this study combines these forms of interactivity with synchronized program code highlighting and visual state updates of data structures.

2.3. Cognitive and Multimedia Foundations of Visualization Design

The effectiveness of algorithm visualization can be further understood through Cognitive Load Theory (Sweller, 1988, 2011) and the Cognitive Theory of Multimedia Learning (Mayer, 2009). These frameworks emphasize that learning with visual representations is influenced by how information is structured and presented, particularly when learners must process multiple sources of information simultaneously.

Cognitive Load Theory distinguishes among intrinsic, extraneous, and germane cognitive load. The inherent complexity of the task determines intrinsic load, while extraneous load arises from the way information is presented (Sweller et al., 1998). In algorithm learning tasks, extraneous load may arise when students must coordinate several elements simultaneously, such as source code, variable values, and changes in data structures. Effective instructional design, therefore, aims to reduce unnecessary cognitive processing while supporting meaningful learning activities.

The Cognitive Theory of Multimedia Learning further specifies design principles that help learners process complex information more efficiently. The signaling principle suggests that highlighting relevant elements can guide learner attention; the segmenting principle supports presenting complex processes in learner-controlled steps; and the contiguity principle emphasizes the coordinated presentation of related information in space or time (Mayer, 2009; Mayer & Chandler, 2001). In algorithm visualizations, these principles correspond to highlighting active variables or matrix elements, allowing step-by-step execution, and synchronizing source code highlighting with changes in the visual representation.

Research on algorithm visualization suggests that designs informed by Cognitive Load Theory and Cognitive Theory of Multimedia Learning are theoretically well supported and have been associated with improved conceptual understanding and engagement in several empirical contexts (Lai et al., 2025; Yusuf & Noor, 2023). For example, in several studies (Végh, 2016a; Végh & Stoffová, 2017; Végh & Takáč, 2017), interactive sorting animations were implemented that aligned with Mayer's multimedia principles, and significant improvements in conceptual understanding were reported. Similarly, Törley (2009) identified the Cognitive Theory of Multimedia Learning as a primary framework guiding the design of algorithm visualization in Hungarian programming education.

These design principles are particularly relevant in matrix-based algorithms, where students must track multiple index positions, intermediate values, and spatial transformations

simultaneously. Visualizations that show active elements, incremental state changes, and the relationship between executed steps may help learners manage this two-dimensional cognitive complexity.

2.4. Reusable Web-Based Visualization Architectures

The last few years have seen a trend towards building web-based visualization systems based on HTML5 and JavaScript. The web-native approach enables platform-independent systems and simplifies the integration of web-based visualization systems into learning management systems (Lee et al., 2025; Liang, 2023). Modern environments build upon classic debugger-based representations of state by combining execution traces and the semantic behavior of programs to generate even richer representations of arrays, in addition to reads, writes, moves, and swaps (Mourato & Santos, 2024). However, a recurring limitation is fragmentation: many tools are created for specific algorithms or data-structure concepts and do not provide reusable architectures or systematic classroom validation (Hansen et al., 2002; Hundhausen et al., 2002; Naps et al., 2003; Shaffer et al., 2010; Vateha & Šimoňák, 2025). Recent examples include visualization systems for lists, stacks, and trees (Mtaho et al., 2024), web-based platforms for teaching data structures (Vélez-Langs & Gaspar-López, 2024), tools for observing runtime behavior, such as garbage collection (Pham et al., 2025), and interactive multimedia environments for algorithm instruction (Tirahna et al., 2024). Ruzsik (2025a) developed a modular JavaScript library for visualizing matrix-based algorithms, demonstrating technical feasibility and architectural extensibility, but without empirical classroom validation. Thus, although reusable architectures have emerged, their systematic pedagogical validation remains limited.

A second way to evaluate algorithm visualization tools from a system design perspective is to examine the types of roles they may provide. Rössling and Freisleben (2002) distinguished four primary roles in algorithm animation systems: the user, the visualizer, the developer, and the programmer. The MatrixVis framework (Ruzsik, 2025b) is designed to support multiple roles simultaneously: it provides an interactive interface for students, a reusable framework for instructors, and a modular structure for developers. However, like most other educational visualization systems, the MatrixVis framework does not perform automatic program-to-visualization mappings for input source code.

2.5. Matrix-Based Algorithm Visualization in Introductory Programming

The empirical literature on matrix-specific algorithm visualization remains relatively limited. Some studies use two-dimensional arrays in contexts such as dynamic programming or graph theory (Geetha et al., 2024; Halim et al., 2012; Lee et al., 2025), and program visualization approaches have also been used to support the implementation of data structures, including matrices (Nathasya et al., 2019). However, most empirical work in algorithm visualization has focused on sorting algorithms and other linear data structures (Grissom et al., 2003; Liang, 2023; Végh, 2016a; Végh & Stoffová, 2017; Végh & Takáč, 2017; Vrachnos & Jimoyiannis, 2014).

Matrix-domain operations involve two-dimensional state propagation and structural transformations that are distinct from linear iteration patterns. Prior work has examined alternative visual models of fundamental data structures, such as matrices, to improve instructional clarity (Végh, 2023). However, systematic empirical investigation of matrix-specific algorithm visualizations in introductory programming contexts remains comparatively limited.

2.6. Research Gap and Novelty of the Present Study

Recent reviews recognize the instructional potential of algorithm visualization while also identifying persistent methodological and application-related gaps (Singh et al., 2024).

Prior studies suggest that visualization can support learning when students actively engage with the system (Fouh et al., 2012; Fu et al., 2025; Grissom et al., 2003; Hundhausen et al., 2002), that stepwise execution, signaling, and synchronized representations are theoretically supported by cognitive and multimedia learning principles (Mayer, 2009; Sweller, 1988), and that web-based systems can improve accessibility and classroom scalability (Lee et al., 2025; Liang, 2023). At the same time, visualization systems remain heterogeneous in design and educational application, making comparison across studies difficult (Sorva, 2012).

Within this literature, four gaps are particularly relevant to the present study. First, most algorithm visualization studies focus on sorting or other one-dimensional structures, and their findings may not directly generalize to matrix-based operations with two-dimensional state propagation, nested indexing, and structural transformations (Shaffer et al., 2010; Végh, 2016a). Second, although modular and technically extensible visualization systems have been developed, reusable JavaScript-based architectures for matrix-oriented algorithm visualization remain relatively uncommon and are often presented primarily as technical systems (Affandi et al., 2016; Halim et al., 2012; Hansen et al., 2002; Lai et al., 2025; Naps et al., 2003; Pareja-Flores et al., 2009; Ruzsik, 2025a; Sibia et al., 2025; Vateha & Šimoňák, 2025). Third, many visualization systems refer to the Cognitive Load Theory or the Multimedia Learning Theory, but concrete features such as step control, highlighting, and synchronized code display are often described functionally rather than through explicit theory-to-design mapping (Affandi et al., 2016; Jurinová & Beňo, 2025; Liang, 2023; Naps et al., 2003; Pareja-Flores et al., 2009; Perháč & Šimoňák, 2022; Vateha & Šimoňák, 2025; Wei et al., 2021). Fourth, although several recent algorithm visualization tools and frameworks have been proposed, their classroom-based validation remains limited. Many studies focus on technical design or perceived usefulness, while fewer examine their use in authentic higher education programming courses with systematic student data (Bende, 2022; Halim et al., 2012; Jurinová & Beňo, 2025; Lee et al., 2025; Liang, 2023; Perháč & Šimoňák, 2022).

The novelty of the present study lies in the combination of four elements. First, it focuses specifically on matrix-based algorithm visualization in introductory programming, thereby extending algorithm visualization research beyond the dominant focus on one-dimensional structures. Second, it evaluates a reusable, modular JavaScript framework originally developed as a technical contribution but not previously subjected to structured classroom evaluation. Third, it provides an explicit theory-to-design mapping between Cognitive Load Theory, Multimedia Learning Theory, usability concepts, and concrete visualization features. Fourth, it provides a systematically documented classroom-based perception study, framed as an initial validation of perceived usability and pedagogical support rather than as evidence of causal learning effects. Figure 1 summarizes the alignment between the theoretical frameworks, the resulting design principles used in the MatrixVis library, and the hypothesized pedagogical outcomes.

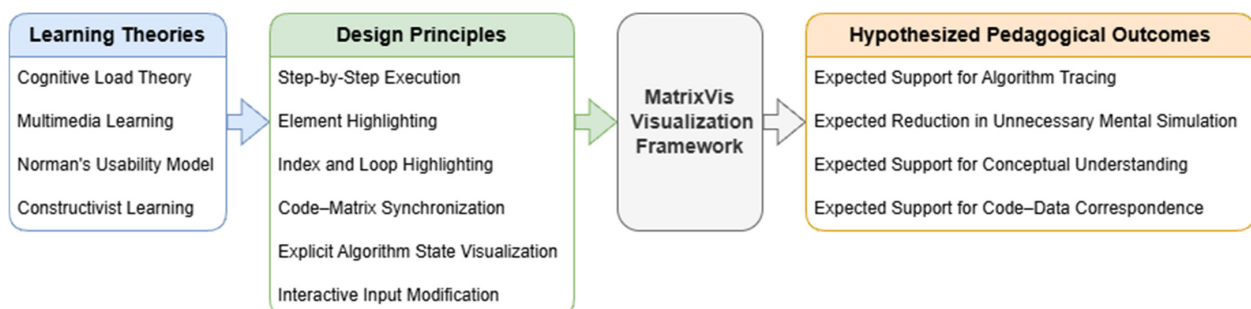


Figure 1. Conceptual alignment between learning theories, implemented design principles, the MatrixVis visualization framework, and hypothesized pedagogical outcomes.

3. The MatrixVis Framework

The MatrixVis JavaScript library serves as the basis for all visualizations used in this paper. The core implementation originated from a master's thesis (Ruzsik, 2025a), extending earlier one-dimensional visualization prototypes (Végh, 2016a, 2016b). The present study focuses on the pedagogical design, classroom implementation, and perception-based evaluation of matrix-based visualizations developed within this framework.

3.1. Architectural Design

The MatrixVis framework (Ruzsik, 2025b) has been implemented as a lightweight, extensible, JavaScript-based system for creating interactive visualizations of simple vector and matrix operations. It allows the seamless integration of interactive visualizations into web-based learning platforms without requiring additional software downloads or hardware dependencies. Because the system runs natively in web browsers, it supports cross-platform use. The architecture of the MatrixVis framework is illustrated in Figure 2.

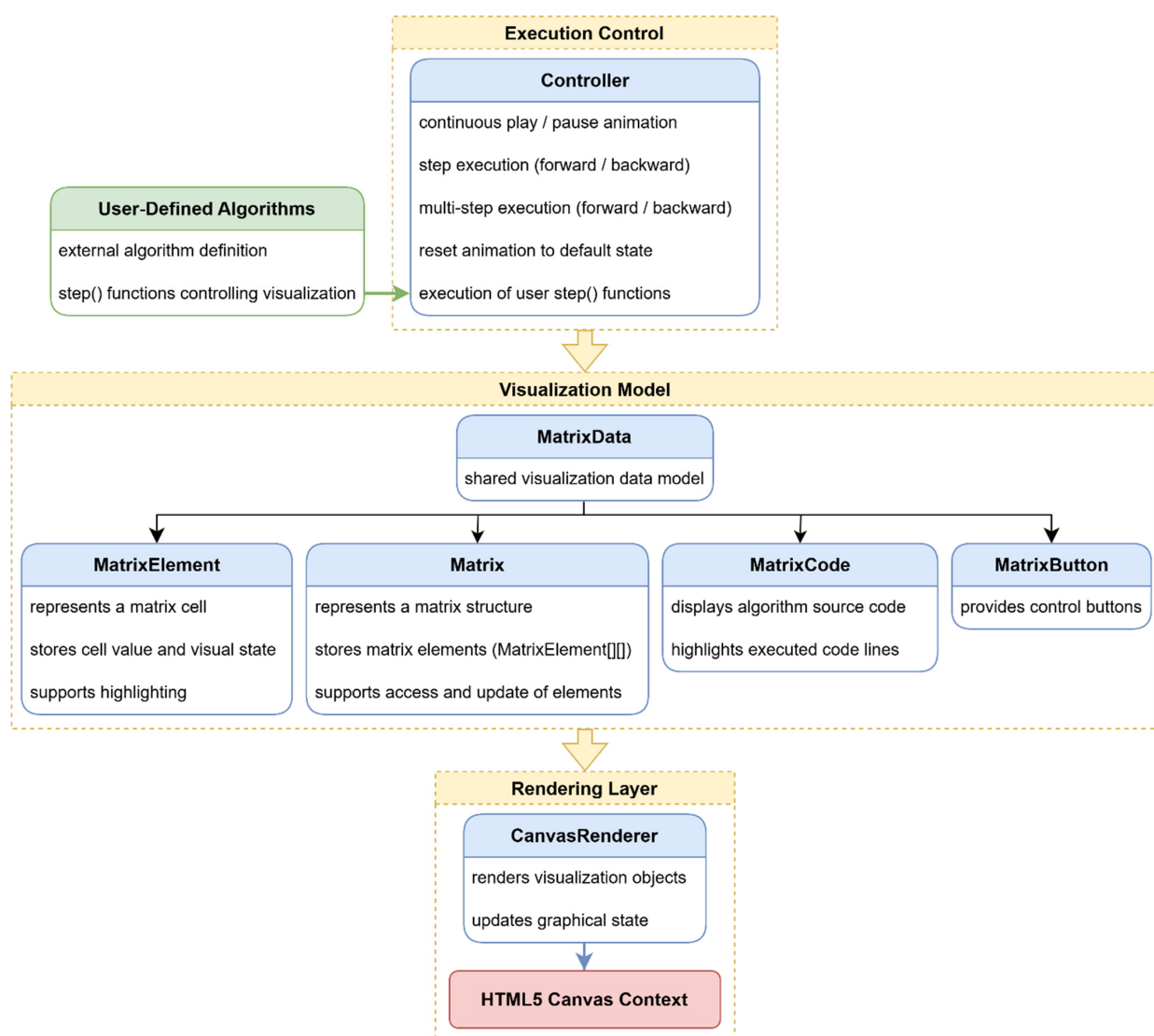


Figure 2. Layered architecture of the MatrixVis visualization framework.

The system consists of three primary layers: the execution control layer, the visualization model, and the rendering layer. In the implementation, these layers are realized through several cooperating classes, including Controller (execution management), Matrix

and MatrixElement (matrix state representation), MatrixCode (code highlighting), MatrixButton (user interaction controls), and CanvasRenderer (visual rendering). Algorithms to be visualized are defined externally through user-implemented `step()` functions, while the framework manages execution flow, visualization state, and graphical rendering.

Within the execution control layer, MatrixVis manages algorithm execution as a sequence of discrete, visually interpretable steps. Instead of executing algorithms as monolithic procedures, they are decomposed into atomic operations (e.g., element comparison, assignment, swap, and index update), each represented as a state transition that can be synchronized with the visualization layer. This modular, step-based execution improves pedagogical transparency by exposing intermediate computational steps and also supports extensibility, as new algorithms can be implemented by defining their execution logic in terms of these basic operations. The execution control layer includes mechanisms that allow users to navigate the visualization step by step, move backward to previously executed states, replay portions of the visualization, or pause it. This step-based structure can support the changing level in the engagement taxonomy proposed by Naps et al. (2003) when learners are allowed to manipulate input values and explore the execution forward and backward.

At the visualization layer, MatrixVis employs dynamic matrix state rendering to represent algorithmic transformations in real time. The visualization model maintains an internal data representation of the current matrix state and continuously synchronizes it with the rendering layer. Whenever an operation modifies the matrix, the rendering module updates the affected elements. Visual highlighting of active indices, compared elements, and modified cells is coordinated with execution steps to reinforce the correspondence between the highlighted source code and structural matrix transformations.

These architectural decisions are aligned with the instructional requirements discussed in Section 2, particularly controlled execution, state visibility, and incremental transformation.

Figure 3 illustrates an example of an algorithm visualization created with the MatrixVis framework, along with its annotated main interface components.

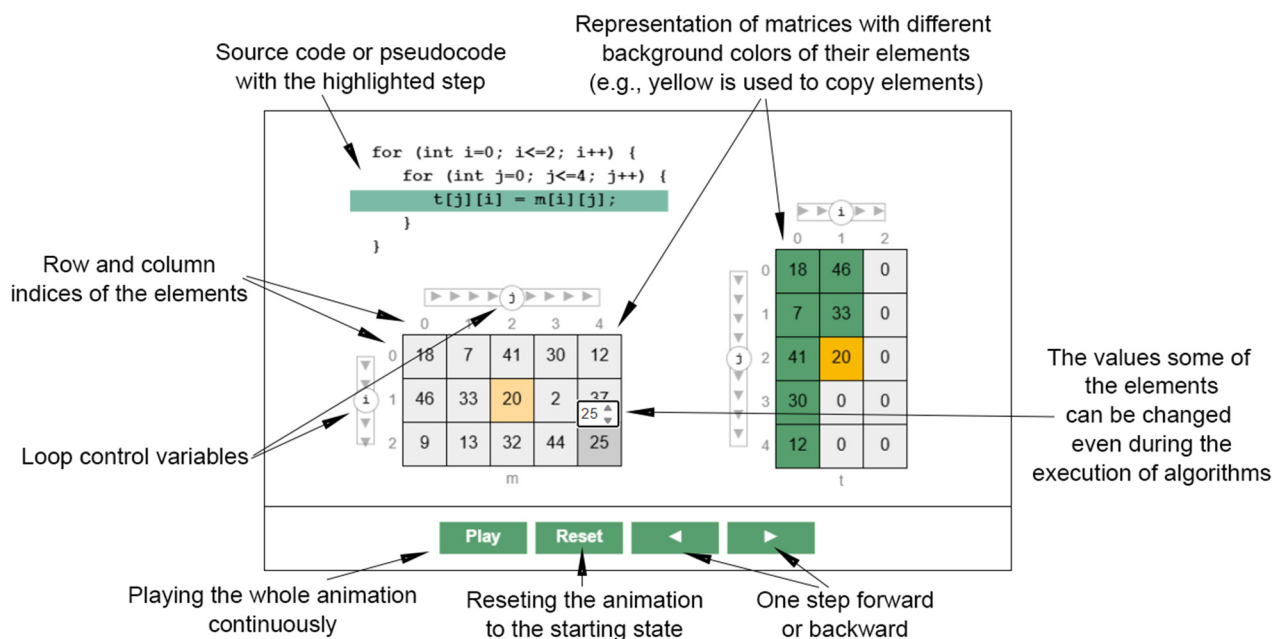


Figure 3. Example of a MatrixVis algorithm visualization with annotated interface components.

3.2. Theory-Informed Design Mapping

Table 1 summarizes the mapping between cognitive principles and their implementations within the MatrixVis framework. This mapping shows a transparent alignment between theory and design, which differentiates the framework from purely technical visualization tools.

Table 1. Theoretical-to-Design Mapping of MatrixVis Features.

Cognitive Principle	MatrixVis Feature
Cognitive Load Theory—Reduction in extraneous load	Visual isolation of active matrix cells and focused presentation of currently relevant elements intended to reduce extraneous cognitive load
Cognitive Load Theory—Learner-controlled pacing	Bidirectional step navigation (forward and backward stepping)
Signaling Principle (Mayer)	Highlighting of the currently processed element (color emphasis)
Segmenting Principle (Mayer)	Manual “next step” control during algorithm animation
Contiguity Principle (Mayer)	Synchronized highlighting of code lines and corresponding matrix operations to maintain spatial and temporal contiguity
Constructivist Learning Orientation	Students can define custom matrix inputs and modify element values during execution, supporting exploratory interaction with algorithm behavior
Immediate Feedback (Inquiry-Based Learning orientation)	Immediate visual feedback after each algorithmic step and after permissible element modification
Gulf of Execution reduction (Norman)	Explicit step-by-step algorithm navigation and controlled manipulation of matrix elements before and during execution, enabling users to perform intended actions directly
Gulf of Evaluation reduction (Norman)	Clear visualization of intermediate algorithm states and the ability to revisit previous execution steps, supporting interpretation, reflection, and error analysis
Transparency of algorithmic state	Persistent display of the matrix, indices, and other relevant variables
Active Learning Facilitation	User-initiated execution control and state revisitation
State consistency preservation	Element editing disabled after algorithmic interaction (comparison, swap, and assignment)

This structured alignment enables critical evaluation of the instructional logic underlying the system. It clarifies how MatrixVis operationalizes selected cognitive, multimedia, usability, constructivist, and inquiry-oriented principles through concrete visualization features, without treating these principles as directly measured outcomes in the present study.

3.3. Implemented Animation Categories

To demonstrate the capabilities of the MatrixVis framework, a set of interactive algorithm visualizations was implemented, covering fundamental operations on both one-dimensional arrays (vectors) and two-dimensional arrays (matrices). These animations were designed to represent common introductory programming tasks that require students to reason about iterative processing, conditional operations, and structural transformations of data structures.

The first group consists of visualizations operating on one-dimensional arrays (Végh, 2025a). These animations illustrate fundamental algorithmic patterns commonly introduced in early programming courses, including aggregation, search, filtering, copying, reversing, and sorting tasks. These vector-based visualizations were primarily used during lectures to support conceptual explanations. However, they were not included in the empirical study of student perceptions reported in this paper.

The second group consists of visualizations operating on two-dimensional arrays (Végh, 2025b). These animations extend the same algorithmic ideas to two-dimensional structures, introducing additional cognitive complexity through nested iteration, row–column indexing, and structural transformations. The implemented matrix-based animations include aggregation operations (e.g., summing matrix elements), conditional evaluation tasks (e.g., counting elements satisfying a condition), element modification procedures, search tasks (e.g., finding the position of a given value or the minimum element), row and column manipulation, structural transformations such as mirroring and transposition, and sorting procedures. These matrix-based algorithm visualizations were presented during the lectures and subsequently used in a student perception survey. The following section describes the methodology used to evaluate students’ perceptions of these visualizations.

4. Methodology

This study investigates students’ perceptions of interactive matrix-based algorithm visualizations (Végh, 2025b) created with the MatrixVis JavaScript library (Ruzsik, 2025b) and demonstrated during introductory programming lectures in April 2025. The methodological design follows a perception-based evaluation approach commonly used in algorithm visualization research, focusing on students’ perceived clarity, usefulness, and instructional support of the visualizations. The empirical investigation was conducted using a structured questionnaire administered after students observed several algorithm visualizations during lectures. The questionnaire collected both quantitative Likert-scale responses and qualitative open-ended reflections. The methodological components of the study are described in the following subsections.

4.1. Study Design and Context

The study was conducted during a university-level introductory programming course in the second semester, following an introductory programming course in the first semester. The course covered sorting one-dimensional arrays using various sorting algorithms, recursion and recursive algorithms, matrix data structures, several basic matrix operations, backtracking techniques, and dynamic data structures. During two lectures focusing on matrix-based algorithms, a set of interactive visualizations (Végh, 2025b) created with the MatrixVis library (Ruzsik, 2025b) was demonstrated to students. The visualizations illustrated typical algorithms for matrix operations, including aggregation, conditional evaluation, search algorithms, element modifications, and structural transformations such as mirroring, transposition, and sorting. During the demonstrations, the algorithms were executed step by step, with the corresponding matrix elements and relevant parts of the source code highlighted. Immediately before the questionnaire was administered, a brief recap of several animations was provided to refresh students’ memories of the visualizations.

The aim of the study was not to compare the effectiveness of different instructional approaches or to establish causal learning effects, but rather to provide an initial perception-based evaluation of the pedagogical usefulness of matrix-based algorithm visualizations in an authentic classroom setting.

4.2. Participants

Participants in the study were first-year undergraduate computer science students enrolled in an introductory programming course in which the visualizations were demonstrated. The total course enrollment was 65 students, including 53 from the Applied Informatics program (47 male, 6 female) and 12 from the Teacher Training in Informatics program (9 male, 3 female). Participation in the survey was voluntary and anonymous.

In total, $N = 48$ responses were collected and included in the analysis, corresponding to a course-enrollment-based response rate of 73.8%. Of these responses, 39 were from students in the Applied Informatics program, and 9 were from students in the Teacher Training in Informatics program.

4.3. Instrument

Data were collected using a structured questionnaire developed to evaluate students' perceptions of the matrix-based algorithm visualizations. The instrument consisted of 14 Likert-scale items (5-point scale: 1 = not at all/very poor, 5 = completely/excellent) and 6 open-ended questions.

The Likert-scale items were organized into four conceptual blocks aligned with the research questions: overall pedagogical evaluation, design-feature-level support, perceived clarity across visualization categories, and perceived value of interactivity and reusability. The questionnaire was partly informed by technology acceptance research, in which perceived usefulness and perceived ease of use are considered central subjective factors in users' acceptance of information systems (Davis, 1989). In the present educational context, these constructs were not used to implement a full Technology Acceptance Model instrument but rather to provide a conceptual rationale for including perception-based items. The items, therefore, focused on students' perceived clarity, usefulness, visual transparency, step-by-step execution, synchronized code highlighting, and the anticipated value of interactive input manipulation.

The open-ended questions allowed students to identify particularly helpful or difficult visualizations and to suggest additional algorithm types for future implementation. These responses were analyzed using inductive coding of the open-ended responses.

The instrument was designed to capture theory-informed design features at the level of perceived pedagogical support rather than as direct cognitive measurements. Accordingly, no standardized instrument for measuring cognitive load was used. Direct cognitive-load assessment would require dedicated measures, such as Paas's nine-point mental-effort rating scale (Paas, 1992) or multi-item instruments distinguishing intrinsic, extraneous, and germane cognitive load (Leppink et al., 2013). Because the aim of the present study was to evaluate students' perceived support after classroom demonstrations rather than to experimentally compare instructional conditions, cognitive load was addressed through a theory-informed design and indirect perception-based evaluation only. The full questionnaire is provided in Appendix A.

4.4. Procedure

The visualizations were demonstrated across two lecture sessions held over a two-week period. The questionnaire was administered in the third week, following a short recap demonstration that refreshed students' memories of the animations. To maintain transparency between the research questions and the measurement design, questionnaire items were grouped into conceptual blocks that reflect the constructs examined in this study. Table 2 summarizes the alignment between the questionnaire items and the research questions.

Table 2. Questionnaire Structure and Research Question Alignment.

Questionnaire Block	Item Numbers	Description
RQ1. Overall pedagogical evaluation	1, 2, 3, 14	Perceived clarity and instructional usefulness
RQ2. Design-feature-level support	4, 5, 6	Signaling (visual highlighting), segmentation (stepwise execution), contiguity (synchronized code display)
RQ3. Differences across visualization categories	7, 8, 9, 10	Aggregation, search, modification, structural transformation
RQ4. Perceived value of interactivity and reusability	11, 12, 13	Perceived support for engagement and independent (self-regulated) learning
Qualitative reflection that supports RQ1–RQ4	15–20	Open-ended reflections on particularly helpful or difficult visualizations and suggestions for additional algorithm categories

5. Results

This section presents the results of the student perception survey. The analysis is organized according to the research questions defined in the Introduction. Because the questionnaire used individual five-point Likert-type items, the analysis was planned with attention to the ordinal nature of Likert-scale responses and the long-standing methodological debate concerning their treatment in statistical analysis (Jamieson, 2004; Sullivan & Artino, 2013). Descriptive statistics were calculated for all questionnaire items. One-sample Wilcoxon signed-rank tests were used to examine whether item ratings differed significantly from the neutral midpoint of the Likert scale (3), while differences among visualization categories were analyzed using the Friedman test.

Before deciding whether item scores should be combined, internal consistency was examined for the conceptual questionnaire blocks. Only the visualization-category block (Q7–Q10) showed good internal consistency (Cronbach's $\alpha = 0.832$). The remaining blocks did not show sufficient internal consistency for scale aggregation: the design-feature block (Q4–Q6) showed low internal consistency (Cronbach's $\alpha = 0.494$), the interactivity and reusability block (Q11–Q13) showed low internal consistency (Cronbach's $\alpha = 0.358$), and the overall pedagogical evaluation block (Q1, Q2, Q3, Q14) was not suitable for aggregation because the corrected item-total correlations were near zero or negative, ranging from -0.208 to 0.064 . Therefore, items from blocks with insufficient internal consistency were analyzed and interpreted at the individual-item level rather than being grouped into composite scores.

5.1. Overall Pedagogical Evaluation (RQ1)

Research Question 1 examined how students evaluated the overall pedagogical usefulness of the interactive matrix-based algorithm visualizations. Four questionnaire items addressed this aspect: the clarity of algorithm explanations (Q1), the perceived usefulness of the animations for understanding the algorithms (Q2), the perceived difficulty of understanding the algorithms without animations (Q3), and the overall instructional support of the visualizations (Q14).

As Table 3 shows, students evaluated the pedagogical usefulness of the visualizations very positively. The clarity of algorithm explanations received a high rating ($M = 4.40$, $SD = 0.79$), similar to the overall instructional support of the visualizations ($M = 4.40$, $SD = 0.77$). The perceived usefulness of the animations for understanding the algorithms obtained the highest score ($M = 4.56$, $SD = 0.65$). One-sample Wilcoxon signed-rank tests indicated that all evaluated items were rated significantly above the scale's neutral

midpoint (3). The strongest evidence against the neutral midpoint was observed for the perceived usefulness of the animations ($Z = 6.058$, $p < 0.001$). Students also indicated that understanding the algorithms without animations would have been more difficult ($M = 3.33$, $SD = 1.02$), which was still significantly above the neutral level.

Table 3. Descriptive statistics and one-sample Wilcoxon signed-rank test results for overall pedagogical evaluation (RQ1).

Item	Mean	Std. Dev.	Z	p
Q1—clarity of explanations	4.40	0.792	5.661	<0.001
Q2—usefulness of animations	4.56	0.649	6.058	<0.001
Q3—difficulty without animations	3.33	1.018	2.220	0.026
Q14—overall instructional support	4.40	0.765	5.804	<0.001

5.2. Design-Feature-Level Support (RQ2)

Research Question 2 examined how specific visualization design features supported students' understanding of the algorithms. Three questionnaire items addressed this aspect: the clarity of the animations' visual presentation (Q4), the usefulness of step-by-step execution (Q5), and the highlighting of the source code during the animations (Q6).

Table 4 summarizes the results. Students evaluated all examined visualization design features very positively. The clarity of the visual presentation of the animations received a high rating ($M = 4.58$, $SD = 0.71$), and the step-by-step execution of the algorithms was also perceived as highly helpful ($M = 4.54$, $SD = 0.58$). The highest rating was obtained for source code highlighting during the animation ($M = 4.69$, $SD = 0.59$). One-sample Wilcoxon signed-rank tests indicated that all evaluated design features were rated significantly above the neutral midpoint of the scale (3), suggesting that students perceived these visualization elements as highly supportive of understanding the visualized algorithms.

Table 4. Descriptive statistics and one-sample Wilcoxon signed-rank test results for design-feature-level support (RQ2).

Item	Mean	Std. Dev.	Z	p
Q4—clarity of visual presentation	4.58	0.710	6.079	<0.001
Q5—step-by-step execution	4.54	0.582	6.120	<0.001
Q6—source code highlighting	4.69	0.589	6.247	<0.001

5.3. Differences Across Visualization Categories (RQ3)

Research Question 3 examined whether students perceived differences in the clarity and comprehensibility of the animations across different categories of matrix-based algorithms. Four questionnaire items addressed this aspect: animations demonstrating aggregation tasks (Q7), search tasks (Q8), modification tasks (Q9), and structural transformations of matrices (Q10). The four items showed good internal consistency, Cronbach's $\alpha = 0.832$. However, because the research question focused on differences between visualization categories, the category ratings were preserved as separate variables and compared using the Friedman test rather than being aggregated into a composite score.

As Table 5 shows, students evaluated the animations' clarity positively across all algorithm categories examined. Animations demonstrating aggregation tasks received the highest rating ($M = 4.58$, $SD = 0.71$), followed by modification tasks ($M = 4.48$, $SD = 0.88$) and search tasks ($M = 4.42$, $SD = 0.85$). Structural transformations received slightly lower ratings ($M = 4.08$, $SD = 1.05$), although they were still evaluated positively. The latter may

reflect the greater conceptual complexity of algorithms with matrix transformations, such as mirroring, transposition, or sorting.

Table 5. Students' evaluation of animation clarity across different algorithm categories.

Item	Mean	Std. Dev.
Q7—aggregation tasks	4.58	0.710
Q8—search tasks	4.42	0.846
Q9—modification tasks	4.48	0.875
Q10—structural transformations	4.08	1.048

A Friedman test revealed a statistically significant difference in perceived clarity across algorithm categories, $\chi^2(3) = 18.579$, $p < 0.001$, Kendall's $W = 0.129$. Bonferroni-corrected post hoc comparisons showed a significant difference between aggregation tasks (Q7) and structural transformations (Q10), adjusted $p = 0.023$. No other pairwise comparisons were statistically significant after correction. Animations demonstrating aggregation tasks received the highest ratings (mean rank = 2.77), while animations illustrating structural transformations received the lowest clarity ratings (mean rank = 2.01). Structural transformations involve more complex spatial changes, which may explain their slightly lower clarity ratings.

5.4. Perceived Value of Interactivity and Reusability (RQ4)

Research Question 4 first examined students' perceptions of the potential value of interactive features in the animations (Q11). During the lecture demonstrations, it was shown that certain matrix elements could be dynamically modified, allowing different input values to be tested. Students were therefore asked to estimate how the possibility of experimenting with such interactive features might support their understanding.

Two additional questionnaire items examined the perceived usefulness of the animations for independent learning. Students indicated that they would likely revisit the animations for self-study (Q12) and that the visualizations could support preparation for homework assignments or examinations (Q13).

Table 6 shows the results. Students indicated that the possibility of experimenting with dynamically modifiable inputs could support their understanding ($M = 3.90$, $SD = 1.08$). Similarly, students reported that they would likely revisit the animations for independent study in the future ($M = 3.90$, $SD = 1.04$). The perceived usefulness of the animations for preparing for homework assignments or examinations received the highest rating among these items ($M = 4.48$, $SD = 0.62$). One-sample Wilcoxon signed-rank tests showed that all three items were rated significantly above the neutral midpoint of the scale (3), indicating that students perceived both the potential interactivity and the availability of the animations for self-study as valuable learning support. These findings suggest that students not only value the visualizations during lectures but also perceive them as useful resources for independent learning and exam preparation.

Table 6. Descriptive statistics and one-sample Wilcoxon signed-rank test results for perceived value of interactivity and reusability (RQ4).

Item	Mean	Std. Dev.	Z	p
Q11—expected usefulness of interactive manipulation	3.90	1.077	4.191	<0.001
Q12—likelihood of revisiting animations for self-study	3.90	1.036	4.417	<0.001
Q13—usefulness for homework or exam preparation	4.48	0.618	6.040	<0.001

5.5. Qualitative Analysis of Open Responses

The questionnaire included six open-ended questions that asked students about their perceptions of helpful and challenging animations and sought suggestions for additional algorithms. The responses were analyzed using inductive coding, in which individual responses were first assigned descriptive codes at the concept level (e.g., minimum, counting, and mirroring) and subsequently grouped into broader thematic categories that reflected algorithm types and perceived characteristics of the visualizations. The coding scheme was developed inductively from the responses and then applied independently by two authors. Because multiple codes could be assigned to a single response, the coding was transformed into binary code-presence variables. For each response and each possible code within the corresponding question, a value of 1 indicated that the coder assigned the code, whereas 0 indicated that the code was not assigned. Cohen's kappa indicated high intercoder agreement for descriptive codes, $\kappa = 0.889$, $SE = 0.016$, $p < 0.001$, and for broader themes, $\kappa = 0.927$, $SE = 0.015$, $p < 0.001$. This high level of intercoder agreement strengthens the dependability of the qualitative coding process and supports the trustworthiness of the resulting thematic structure. Discrepancies and ambiguous cases were discussed and resolved through consensus before the final theme frequencies were calculated. The resulting broader themes and their frequencies are summarized in Table 7.

Aggregation tasks (Q15): As Table 7 shows, specific aggregation task animations were identified as helpful in 15 coded theme mentions, while 14 theme mentions contained general positive comments about aggregation task animations. The specific operations mentioned included determining the minimum or maximum value (8 mentions), counting elements satisfying a condition (5 mentions), and summing matrix elements (4 mentions). These comments correspond with the quantitative findings reported in Section 5.3, where aggregation-related animations received the highest clarity ratings among the evaluated algorithm categories.

Search tasks (Q16): Search-related visualizations received 16 coded theme mentions containing general positive comments and 13 theme mentions in which a specific search task animation was identified as helpful. The specific examples mentioned included determining the position of a given value (10 mentions) and locating the position of the minimum element in the matrix (5 mentions). These responses suggest that the visualizations helped students follow how index variables changed during the execution and how conditional checks influenced the search process.

Modification tasks (Q17): As Table 7 shows, specific modification task animations were identified as helpful in 19 coded theme mentions, while 9 coded mentions contained general positive comments about modification task animations. Three responses explicitly indicated that no modification task animation was found to be particularly helpful. The specific operations mentioned included swapping columns (16 mentions) or rows (11 mentions), modifying all elements of a given row (2 mentions) or column (2 mentions), and setting the values of a row or column to zero (2 mentions). These responses suggest that the step-by-step visualizations helped students observe how assignment operations and structural updates directly changed the matrix's state.

Structural transformations (Q18) and perceived difficulty (Q19): Specific structural transformation animations were identified as helpful in 26 coded theme mentions, while 8 coded mentions contained general positive comments about structural transformation animations. The examples mentioned included mirroring (14 mentions), transposition (10 mentions), sorting (6 mentions), and sorting via a one-dimensional auxiliary array (1 mention). At the same time, Q19 showed that difficulties were most often related to structural transformation animations, with 20 coded theme mentions referring to this type of difficulty. In contrast, 16 responses explicitly stated that no animation was difficult to follow. At

the operation level, the most frequently mentioned difficult operation was in-place matrix sorting (10 mentions), followed by mirroring (5 mentions), sorting in general (3 mentions), sorting via a one-dimensional auxiliary array (3 mentions), and transposition (2 mentions). In addition, one mention referred to difficulty following the source code, and one mention referred to difficulty related to prior mathematical knowledge. The increased difficulty may be attributed to more complex spatial changes and the need to modify multiple matrix elements simultaneously during algorithm execution. These qualitative observations align with the quantitative findings in Section 5.3, where visualizations of algorithms involving structural transformations received slightly lower clarity ratings than visualizations of other algorithm categories.

Table 7. Frequency counts of broader themes identified in the open-ended responses.

Item	Broader Theme	Frequency
Q15	A specific aggregation task animation was helpful	15
Q15	General positive comment about aggregation task animations	14
Q15	None of the aggregation task animations was identified as helpful	3
Q15	Response referred to a different animation category	1
Q16	General positive comment about search task animations	16
Q16	A specific search task animation was helpful	13
Q16	None of the search task animations was identified as helpful	1
Q16	Response referred to a different animation category	1
Q17	A specific modification task animation was helpful	19
Q17	General positive comment about modification task animations	9
Q17	None of the modification task animations was identified as helpful	3
Q17	Response referred to a different animation category	1
Q18	A specific structural transformation task animation was helpful	26
Q18	General positive comment about transformation task animations	8
Q18	None of the transformation task animations was identified as helpful	1
Q18	Response referred to a different animation category	1
Q19	Difficulty was related to structural transformation task animations	20
Q19	Explicitly stated that no animation was difficult to follow	16
Q19	General difficulty in following the animations was mentioned	2
Q19	Response referred to a task that is not related to these animations	2
Q19	Difficulty was related to the source code of the algorithm	1
Q19	Difficulty was related to prior mathematical knowledge	1
Q20	A general suggestion was made to extend the set of animations	13
Q20	Explicitly stated no additional suggestion	5
Q20	An additional matrix algorithm animation was suggested	3
Q20	An additional programming topic was suggested for visualization	3
Q20	An additional search algorithm animation was suggested	3
Q20	An additional sorting algorithm animation was suggested	2
Q20	Another data-structure topic was suggested for visualization	2
Q20	Another algorithmic topic was suggested for visualization	1

Note. Frequency refers to coded theme mentions rather than mutually exclusive respondent categories because a single open-ended response could contain more than one idea.

Suggestions for additional algorithms (Q20): Students also proposed several algorithms or topics that could be helpful if they were visualized in a similar way. The most frequent theme in the suggestions was a general request to extend the set of available animations (13 mentions). Five responses explicitly stated that there was no additional algorithm or topic to suggest. More specific suggestions included additional matrix algorithms (3 mentions), programming topics (3 mentions), search algorithms (3 mentions), sorting algorithms (2 mentions), data-structure topics (2 mentions), and other algorithmic topics (1 mention). Examples included computing the determinant of a matrix, working with triangular matrices, binary search, data structures, and comparing similar algorithms. Such suggestions indicate that students perceived algorithm visualizations not only as helpful instructional aids during lectures but also as tools to support the exploration of more advanced algorithmic concepts.

Overall, the qualitative student feedback added to the quantitative survey results, providing further insight into how students felt about the visualizations. The responses suggest that structural transformation animations were pedagogically valuable but also cognitively more demanding. The suggestions for additional topics indicate that students saw potential for extending the interactive visualization approach beyond the demonstrated matrix algorithms.

6. Discussion

The results of the perception study provide several insights regarding the pedagogical role of matrix-based algorithm visualizations and their alignment with established visualization and multimedia learning theories.

6.1. Theory–Design Alignment

The results provide perception-based indications that the theory-informed design principles implemented in the MatrixVis framework were supportive of students. Respondents evaluated the clarity of the visual presentation, the step-by-step execution, and the synchronized code highlighting positively, with all design-feature items receiving mean ratings above the neutral midpoint. These findings are consistent with theoretical expectations derived from Cognitive Load Theory and the Cognitive Theory of Multimedia Learning. However, they should not be interpreted as direct evidence of reduced cognitive load, improved schema construction, or improved learning outcomes. The positive evaluations of stepwise execution, visual highlighting, and synchronized code display therefore provide perception-level support for the plausibility of the theory-to-design mapping presented in Section 3.2.

These results are compatible with the argument that the effectiveness of algorithm visualization depends not merely on the animation itself but on the cognitive alignment between visual representation and algorithmic logic (Tversky et al., 2002), while remaining limited to perception-level evidence rather than empirical confirmation of the underlying cognitive mechanisms.

6.2. Differences in Perceived Difficulty Across Algorithm Categories

Visualizations demonstrating aggregation tasks received the highest clarity ratings, whereas structural transformation animations received lower ratings. These findings may be interpreted in light of theoretical expectations concerning intrinsic cognitive load. Most aggregation tasks require only a linear traversal and simple state transitions. In contrast, structural transformations such as mirroring, transposition, or sorting require learners to track spatial relationships and multiple coordinated changes in the matrix. These characteristics are expected to increase element interactivity and, therefore, intrinsic cognitive load

(Sweller, 2011). Similar observations have been reported in previous research on algorithm visualization, where tasks involving complex structural manipulation required greater cognitive effort from novice learners (Shaffer et al., 2010; Vrachnos & Jimoyiannis, 2014).

Importantly, even the lowest-rated category still received positive evaluations, suggesting that the visualizations were perceived as supporting understanding despite the increased complexity of these algorithms. However, because the present study did not directly measure cognitive load or objective learning performance, this interpretation should be treated as theoretically grounded rather than empirically confirmed.

6.3. Perceived Value of Interactivity

Another important finding concerns the perceived value of interactivity. Students reported that the possibility of experimenting with dynamically modifiable inputs would likely support their understanding and that they would revisit the animations for independent study. These responses correspond with the engagement taxonomy proposed by Naps et al. (2003), which distinguishes between passive viewing and more active forms of interaction, such as responding or changing. While the present study primarily involved instructor-led demonstrations, students' responses suggest that they perceive value in moving toward higher engagement levels. In this sense, allowing learners to modify input values, test alternative cases, and observe algorithm behavior dynamically represents a promising extension of the current instructor-led use of the visualizations. The MatrixVis library used in this study provides a technical basis for future implementations of visualizations that support more interactive forms of algorithm exploration.

6.4. Contributions to Algorithm Visualization Research

The study contributes to algorithm visualization research by combining a matrix-specific instructional domain, a reusable JavaScript-based visualization framework, explicit theory-to-design mapping, and structured classroom-based perception data. First, it extends research on algorithm visualization beyond the dominant focus on sorting algorithms and one-dimensional data structures by examining matrix-based operations in introductory programming. Second, it illustrates how a reusable visualization framework can be evaluated in an authentic classroom context rather than only being described as a technical system. Third, it provides perception-based evidence about how students experienced the clarity, usefulness, and interaction potential of the visualizations. These contributions should be interpreted as a preliminary validation layer rather than as evidence of causal learning effects.

6.5. Pedagogical and Design Implications

The findings of the present study suggest several implications for both instructional practice and future research on algorithm visualization in introductory programming education. From a pedagogical perspective, the results highlight the importance of transparent animation logic. Students evaluated the step-by-step execution and synchronized code highlighting very positively, suggesting that visualizations are particularly useful when they explicitly expose the relationship between program code and changes in the underlying data structures.

The results also suggest that opportunities for learner manipulation may further increase the pedagogical value of algorithm visualizations. Although the visualizations in the present study were primarily used in instructor-led demonstrations, students indicated that the ability to modify input values and experiment with algorithm behavior would likely enhance engagement and support independent study.

From a design perspective, the findings support the value of consistent code–data correspondence, stepwise execution, and visual highlighting as central design elements of

matrix-based algorithm visualizations. However, because the present study was perception-based, these design implications should be treated as hypotheses for future controlled studies rather than as evidence of improved learning outcomes.

7. Limitations

Although the findings show useful insights into students' perceptions of matrix-based algorithm visualizations, several limitations must be considered when interpreting the results.

Firstly, the study primarily examined students' perceptions of the instructional usefulness and clarity of the visualizations rather than directly measuring learning outcomes. It did not include pre- and post-tests, standardized coding assignments, performance-based assessments, a control group, or a comparison with alternative instructional approaches. Therefore, the findings do not allow conclusions about measurable improvements in algorithmic understanding. Furthermore, although the visualizations were designed based on principles derived from Cognitive Load Theory, the study did not include direct measurement of cognitive load with validated instruments. As a result, the findings provide only indirect, perception-based evidence regarding cognitive processing. Future research could incorporate algorithm-tracing tasks, problem-solving exercises, standardized coding assignments, or controlled pedagogical experiments comparing visualization-based instruction with traditional instruction.

Secondly, the study was conducted within a single course at a single university, and the sample comprised students enrolled in an introductory programming course. Although the participants represented the authentic target audience for the presented visualizations, the findings may not be fully generalizable to other institutional contexts, programming curricula, or student populations. Replication studies across different institutions and programming courses would therefore be valuable for confirming the broader applicability of the observed patterns.

Thirdly, the course-based evaluation context may have introduced demand characteristics and instructor-as-evaluator effects. The visualizations were demonstrated by the course instructor, and the questionnaire was administered shortly after a recap session led by the instructor who presented the visualizations. Although participation was voluntary and anonymous, students may have assumed that positive evaluations were expected or may have been hesitant to express critical views. Future studies should reduce this potential bias by using independent researchers to administer the evaluation, clearly separating data collection from regular instruction, delaying questionnaire administration, and combining perception-based measures with objective performance assessments.

In addition, although students were able to observe the dynamic execution of the algorithms and follow step-by-step animations, they did not directly interact with the system during the lectures. As indicated by the questionnaire responses, learners expressed interest in experimenting with the visualizations by modifying input values and exploring alternative behaviors of the algorithms. Future studies could therefore investigate the pedagogical effects of fully interactive visualization environments that allow learners to actively manipulate algorithm inputs and observe the resulting program states.

Finally, the qualitative analysis relied on inductive coding of relatively short open-ended responses. While it provided useful insights, more extensive qualitative data could provide a deeper understanding of how learners interpret and use algorithm visualizations.

8. Conclusions

This perception study examined the pedagogical potential of interactive matrix-based algorithm visualizations implemented with the MatrixVis JavaScript library. The paper

contributes to the algorithm visualization literature by focusing on algorithms operating on two-dimensional arrays. This data structure has received less attention in prior research compared to traditional data structures such as vectors, lists, or trees.

The findings indicate that students perceive stepwise execution, synchronized code highlighting, and visually explicit data transformations as particularly helpful for understanding the logic of the presented algorithms. These results support the idea that visualization designs grounded in established cognitive and multimedia learning principles may meaningfully support novice programmers' understanding of algorithm behavior.

Beyond the specific classroom context of the present study, the results also highlight the potential value of reusable web-based visualization architectures for programming education. Framework-based approaches, such as the MatrixVis JavaScript library presented in this paper, may enable instructors to create consistent visualization environments across a wider range of algorithmic concepts.

Future work may expand the present study by examining fully interactive learning scenarios and by investigating the effects of visualization-based instruction on quantifiable measures of students' programming performance.

In conclusion, the results indicate that carefully developed algorithm visualizations, particularly visualizations with clear code–data correspondence and structured stepwise execution support, represent a promising direction for improving the teaching and learning of algorithmic thinking in introductory programming courses.

Author Contributions: Conceptualization, L.V. and O.T.; methodology, L.V. and O.T.; software, L.V.; validation, L.V., O.T. and D.D.; formal analysis, L.V. and D.D.; investigation, L.V.; resources, L.V.; data curation, L.V.; writing—original draft preparation, L.V. and D.D.; writing—review and editing, O.T. and K.C.; visualization, L.V. and D.D.; supervision, L.V.; project administration, L.V. and K.C.; funding acquisition, K.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Education, Science, Research and Sport of the Slovak Republic, grant number KEGA 014TTU-4/2024 (“Intelligent Animation-Simulation Models, Tools, and Environments for Deep Learning”) and KEGA 005UJS-4/2026 (“Integration of Visualization and Modeling Strategies into the Higher Education Training of Science and Informatics Teachers”). The APC was funded by KEGA 014TTU-4/2024.

Institutional Review Board Statement: Ethical review and approval were not required for this study because it involved the voluntary participation of university students, the collection of fully anonymous questionnaire data, and no personally identifiable or sensitive data were collected. No intervention was involved, and the study was conducted in accordance with the Ethical Code of J. Selye University and generally accepted ethical standards for research involving human participants.

Informed Consent Statement: Verbal informed consent was obtained from all participants prior to questionnaire completion. Verbal consent was used because the study involved an anonymous and voluntary educational questionnaire in which no personal or sensitive data were collected. Participants were informed that participation was voluntary and anonymous and that they could decline participation by choosing not to complete the questionnaire.

Data Availability Statement: The dataset generated and analyzed during the current study, consisting of anonymous student questionnaire responses, is openly available in Zenodo at <https://doi.org/10.5281/zenodo.20381954>.

Acknowledgments: The authors would like to thank Márton László Ruzsik for implementing the MatrixVis framework as part of his master's thesis (Ruzsik, 2025a), which provided the technical foundation for the visualizations used in this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Full Questionnaire (English Version)

Questionnaire for the Evaluation of Educational Algorithm Visualizations

This questionnaire concerns the matrix-based algorithm visualizations presented during the previous two lectures. At the beginning of the session, several examples were briefly revisited. Please respond based on what you remember or what you were able to recall during the demonstration.

Please use the following response scale:

1 = not at all/very poor

5 = completely/excellent

General Impression

1. How understandable were the explanations of the algorithms presented during the lectures?
(1 = not understandable at all, 5 = completely understandable)
2. To what extent did the animations help you understand the algorithms?
(1 = did not help at all, 5 = helped very much)
3. If there had been no animations, how difficult would it have been to follow the algorithms?
(1 = not difficult at all, 5 = very difficult)

Overall Evaluation of the Animations

4. How clear and illustrative was the visual presentation of the animations?
(e.g., display of index variables, highlighting of elements in orange during comparison and swapping)
(1 = confusing, 5 = clear and easy to follow)
5. To what extent did the step-by-step execution support your understanding?
(1 = not at all, 5 = very helpful)
6. To what extent did the highlighting of the source code during the animation help your understanding?
(1 = not at all, 5 = very helpful)

Perception by Type of Animation

7. How clear and easy to follow were the animations demonstrating aggregation tasks (e.g., summing elements, counting, determining the minimum)?
(1 = not clear, 5 = completely clear)
8. How clear and easy to follow were the animations demonstrating search tasks (e.g., determining the indices of a given value, finding the position of the minimum)?
(1 = difficult to follow, 5 = completely clear)
9. How understandable were the animations involving modification tasks (e.g., multiplying elements by two, setting all elements of a row or column to zero, swapping rows or columns)?
(1 = not understandable, 5 = completely understandable)
10. How easy to follow were the animations demonstrating structural transformations (e.g., mirroring, transposition, sorting)?
(1 = not easy to follow, 5 = completely easy to follow)

Usability and Learning Intentions

11. During the lecture, it was demonstrated that certain elements could be modified dynamically (e.g., entering custom input values). If you had been able to experiment with these animations yourself, to what extent do you think this would have supported

your understanding?

(1 = not at all, 5 = very much)

12. How likely is it that you would rewatch these animations online for independent study in the future?
(1 = definitely not, 5 = definitely yes)
13. To what extent do you think these animations would support you when preparing homework assignments or exams?
(1 = not at all, 5 = very helpful)
14. Overall, to what extent do you feel that the animations supported your understanding of matrix-based algorithms?
(1 = did not support, 5 = supported very much)

Open-Ended Questions

Please answer the following questions briefly in writing.

15. Was there any animation demonstrating aggregation tasks (e.g., summing elements, counting, determining the minimum) that you found particularly helpful? If yes, which one?
16. Was there any animation demonstrating search tasks (e.g., determining the indices of a given value, finding the position of the minimum) that particularly supported your understanding?
17. Was there any animation involving modification tasks (e.g., multiplying elements by two, setting all elements of a row or column to zero, swapping rows or columns) that you found particularly helpful? If yes, which one?
18. Was there any animation demonstrating structural transformations (e.g., mirroring, transposition, sorting) that particularly supported your understanding? If yes, which one?
19. Was there any animation that you found difficult to follow? If yes, which one and why?
20. What additional algorithms or topics would you like to see demonstrated using similar animations?

References

- Affandi, A., Shiratuddin, N., & Bahrin, S. (2016). Critical analysis on algorithm visualization study. *International Journal of Computer Applications*, 150(11), 18–22. [\[CrossRef\]](#)
- Bende, I. (2022). A case study of the integration of algorithm visualizations in Hungarian programming education. *Teaching Mathematics and Computer Science*, 20(1), 51–66. [\[CrossRef\]](#)
- Brown, M. H., & Sedgewick, R. (1984). A system for algorithm animation. In *Proceedings of the 11th annual conference on computer graphics and interactive techniques, SIGGRAPH '84, Minneapolis, MA, USA, July 23–27* (pp. 177–186). ACM. [\[CrossRef\]](#)
- Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3), 319–340. [\[CrossRef\]](#) [\[PubMed\]](#)
- Fouh, E., Akbar, M., & Shaffer, C. A. (2012). The role of visualization in computer science education. *Computers in the Schools*, 29(1–2), 95–117. [\[CrossRef\]](#)
- Fu, Q., Zhou, X., Zheng, Y., & Wang, Z. (2025). The effects of algorithm visualisation on elementary school students' algorithm-learning performance, motivation, and behaviour. *Journal of Computer Assisted Learning*, 41(3), e70049. [\[CrossRef\]](#)
- Geetha, S., K, S., V, M., Joshi, A. S., & C S, A. (2024). Animated algorithm visualizer for graph based algorithms and recursive programs using machine learning. In *2024 International conference on intelligent and innovative technologies in computing, electrical and electronics (IITCEE), Bangalore, India, January 24–25* (pp. 1–5). IEEE. [\[CrossRef\]](#)
- Grissom, S., McNally, M. F., & Naps, T. (2003). Algorithm visualization in CS education: Comparing levels of student engagement. In *Proceedings of the 2003 ACM symposium on software visualization, San Diego, CA, USA, June 11–13* (pp. 87–94). ACM. [\[CrossRef\]](#)
- Halim, S., Koh, Z. C., Loh, V. B. H., & Halim, F. (2012). Learning algorithms with unified and interactive web-based visualization. *Olympiads in Informatics*, 6, 53–68.
- Hansen, S., Narayanan, N. H., & Hegarty, M. (2002). Designing educationally effective algorithm visualizations. *Journal of Visual Languages & Computing*, 13(3), 291–317. [\[CrossRef\]](#)

- Hmelo-Silver, C. E., Duncan, R. G., & Chinn, C. A. (2007). Scaffolding and achievement in problem-based and inquiry learning: A response to Kirschner, Sweller, and Clark (2006). *Educational Psychologist*, 42(2), 99–107. [\[CrossRef\]](#)
- Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3), 259–290. [\[CrossRef\]](#)
- Jamieson, S. (2004). Likert scales: How to (ab)use them. *Medical Education*, 38(12), 1217–1218. [\[CrossRef\]](#) [\[PubMed\]](#)
- Jurinová, J., & Beňo, M. (2025). Web application for the development of algorithmic thinking and programming skills: Synergistic integration of code visualisation, memory model, and doubly linked list in C. In *2025 International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Stry Smokovec, Slovakia, November 13–14 (pp. 299–307). IEEE. [\[CrossRef\]](#)
- Lai, C.-H., Ho, L.-C., & Liao, Z.-Y. (2025). Sustainability of programming education through CDIO-oriented practice: An empirical study on syntax-level structural visualization for functional programming languages. *Sustainability*, 17(12), 5630. [\[CrossRef\]](#)
- Lee, D. H., Prasad, A., Deo-Campo Vuong, R., Wang, T., Han, E., & Kempe, D. (2025). dpvis: A visual and interactive learning tool for dynamic programming. In *Proceedings of the 56th ACM technical symposium on computer science education V. 1*, Pittsburgh PA USA, February 26–March 1 (pp. 645–651). ACM Conferences. [\[CrossRef\]](#)
- Leppink, J., Paas, F., Van Der Vleuten, C. P. M., Van Gog, T., & Van Merriënboer, J. J. G. (2013). Development of an instrument for measuring different types of cognitive load. *Behavior Research Methods*, 45(4), 1058–1072. [\[CrossRef\]](#) [\[PubMed\]](#)
- Liang, Y. D. (2023). Teaching and learning sorting algorithms using animation. In *2023 international conference on computational science and computational intelligence (CSCI)*, Las Vegas, NV, USA, December 13–15 (pp. 1755–1759). IEEE. [\[CrossRef\]](#)
- Liu, J., Poulsen, S., Goodwin, E., Chen, H., Williams, G., Gertner, Y., & Franklin, D. (2024). Teaching algorithm design: A literature review. *arXiv*. [\[CrossRef\]](#)
- Mayer, R. E. (2009). *Multimedia learning* (2nd ed.). Cambridge University Press. [\[CrossRef\]](#)
- Mayer, R. E. (2024). The past, present, and future of the cognitive theory of multimedia learning. *Educational Psychology Review*, 36(1), 8. [\[CrossRef\]](#)
- Mayer, R. E., & Chandler, P. (2001). When learning is just a click away: Does simple user interaction foster deeper understanding of multimedia messages? *Journal of Educational Psychology*, 93(2), 390–397. [\[CrossRef\]](#)
- Mourato, R., & Santos, A. L. (2024). Educational program visualizations using synthesized execution information. *OASICs, ICPEC 2024*, 122, 10:1–10:8. [\[CrossRef\]](#)
- Mtaho, A. B., Masoud, M. M., & Mselle, L. J. (2024). Developing a celiotm programming learning tool to facilitate teaching and learning data structure concepts in C++ for novice programmers. *Information Technologies and Learning Tools*, 101(3), 42–70. [\[CrossRef\]](#)
- Myller, N., Bednarik, R., Sutinen, E., & Ben-Ari, M. (2009). Extending the engagement taxonomy: Software visualization and collaborative learning. *ACM Transactions on Computing Education*, 9(1), 1–27. [\[CrossRef\]](#)
- Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. Á. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, 35(2), 131–152. [\[CrossRef\]](#)
- Nathasya, R. A., Karnalim, O., & Ayub, M. (2019). Integrating program and algorithm visualisation for learning data structure implementation. *Egyptian Informatics Journal*, 20(3), 193–204. [\[CrossRef\]](#)
- Norman, D. A. (2013). *The design of everyday things* (Rev. and expanded edition). MIT Press. (Original work published 1988).
- Paas, F. G. W. C. (1992). Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. *Journal of Educational Psychology*, 84(4), 429–434. [\[CrossRef\]](#)
- Pareja-Flores, C., Urquiza-Fuentes, J., & Velázquez Iturbide, J. Á. (2009). Web-based algorithm and program visualization for education. In D. B. A. Mehdi Khosrow-Pour (Ed.), *Encyclopedia of information science and technology* (2nd ed., pp. 4093–4098). IGI Global. [\[CrossRef\]](#)
- Perháč, P., & Šimoňák, S. (2022). Interactive system for algorithm and data structure visualization. *Computer Science Journal of Moldova*, 30(88), 28–48. [\[CrossRef\]](#)
- Pham, M. L., Stallings, J., Mayor, P. O., & Schreib, R. (2025). An interactive visualization tool for teaching garbage collection. In *Proceedings of the 30th ACM conference on innovation and technology in computer science education V. 1*, Nijmegen, The Netherlands, June 27–July 2 (pp. 667–673). ACM. [\[CrossRef\]](#)
- Prasad, A., Chaudhary, K., & Sharma, B. (2021). Programming skills: Visualization, interaction, home language and problem solving. *Education and Information Technologies*, 27(3), 3197–3223. [\[CrossRef\]](#)
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172. [\[CrossRef\]](#)
- Rössling, G., & Freisleben, B. (2002). ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages & Computing*, 13(3), 341–354. [\[CrossRef\]](#)
- Ruzsik, M. L. (2025a). *Javascript knižnica na tvorbu interaktívnych vizualizácií algoritmov používajúce matice—Javascript könyvtár mátrixokat használó algoritmusok interaktív vizualizációinak készítésére* [Master's thesis, J. Selye University].
- Ruzsik, M. L. (2025b, March). *Matrix-based algorithm visualizations (matrixvis) javascript library* [JavaScript]. Available online: <https://github.com/RuzsikMarton/javascript-library-matrixalgorithm> (accessed on 25 February 2026).

- Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010). Algorithm visualization: The state of the field. *ACM Transactions on Computing Education*, 10(3), 1–22. [CrossRef]
- Sibia, N., Liut, M., & Nobre, C. (2025). *Exploring the role of visualization tools in enhancing computing education: A systematic literature review* [EuroVis 2025—Education Papers]. The Eurographics Association. [CrossRef]
- Singh, V., Dubey, S., & Ahmad, E. A. (2024). Visualizing algorithms in the field of education: A comprehensive review. *International Journal for Research in Applied Science and Engineering Technology*, 12(2), 1670–1675. [CrossRef]
- Sorva, J. (2012). *Visual program simulation in introductory programming education*. Aalto University School of Science.
- Sullivan, G. M., & Artino, A. R. (2013). Analyzing and interpreting data from likert-type scales. *Journal of Graduate Medical Education*, 5(4), 541–542. [CrossRef] [PubMed]
- Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2), 257–285. [CrossRef] [PubMed]
- Sweller, J. (2011). Cognitive load theory. In *Psychology of learning and motivation* (Vol. 55, pp. 37–76). Elsevier. [CrossRef]
- Sweller, J., Van Merriënboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, 10(3), 251–296. [CrossRef]
- Tirahna, A. Z., Mursid, R., & Sriadhi, S. (2024). Development of a web-based interactive multimedia problem-based learning approach on algorithms and programming material. *Randwick International of Education and Linguistics Science Journal*, 5(2), 595–611. [CrossRef]
- Törley, G. (2009). Algoritmus-vizualizáció a programozásoktatásban. *Journal of Applied Multimedia*, 3(IV), 81–94.
- Tversky, B., Morrison, J. B., & Betrancourt, M. (2002). Animation: Can it facilitate? *International Journal of Human-Computer Studies*, 57(4), 247–262. [CrossRef]
- Vateha, F., & Šimoňák, S. (2025). Comparative visualization of algorithms and data structures. *Computing and Informatics*, 44(2), 336–365. [CrossRef]
- Velázquez-Iturbide, J. Á. (2025). Reflections on teaching algorithm courses. In *Proceedings of the 56th ACM technical symposium on computer science education V. 1, Pittsburgh, PA, USA, February 26–March 1* (pp. 1148–1154). ACM Conferences. [CrossRef]
- Végh, L. (2016a). JavaScript library for developing interactive micro-level animations for teaching and learning algorithms on one-dimensional arrays. *Acta Didactica Napocensia*, 9(2), 23–32.
- Végh, L. (2016b). *Sorting algorithms*. Available online: <https://ani.ide.sk/en/> (accessed on 10 April 2026).
- Végh, L. (2023). Models of data structures in educational visualizations for supporting teaching and learning algorithms and computer programming. *International Journal of Advanced Natural Sciences and Engineering Researches*, 7(5), 147–154. [CrossRef]
- Végh, L. (2025a, August). *Operations on one-dimensional arrays (vectors)*. Available online: <https://vektor.ide.sk/en/> (accessed on 10 May 2026).
- Végh, L. (2025b, March). *Operations on two-dimensional arrays (matrices)*. Available online: <https://matrix.ide.sk/en/> (accessed on 10 May 2026).
- Végh, L., & Stoffová, V. (2016). An interactive animation for learning sorting algorithms: How students reduced the number of comparisons in a sorting algorithm by playing a didactic game. *Teaching Mathematics and Computer Science*, 14(1), 45–62. [CrossRef]
- Végh, L., & Stoffová, V. (2017). Algorithm animations for teaching and learning the main ideas of basic sortings. *Informatics in Education*, 16(1), 121–140. [CrossRef]
- Végh, L., & Takáč, O. (2017). Using interactive card animations for understanding of the essential aspects of non-recursive sorting algorithms. In J. Janech, J. Kostolny, & T. Gratkowski (Eds.), *Proceedings of the 2015 federated conference on software development and object technologies* (Vol. 511, pp. 336–347). Springer International Publishing. [CrossRef]
- Vélez-Langs, O., & Gaspar-López, J. (2024, December 30). *Enhancing programming education through algorithm visualization: A focus on teaching data structures*. HCI International 2024, Washington DC, USA. [CrossRef]
- Vrachnos, E., & Jimoyiannis, A. (2014). Design and evaluation of a web-based dynamic algorithm visualization environment for novices. *Procedia Computer Science*, 27, 229–239. [CrossRef]
- Wei, X., Lin, L., Meng, N., Tan, W., Kong, S.-C., & Kinshuk. (2021). The effectiveness of partial pair programming on elementary school students' computational Thinking skills and self-efficacy. *Computers & Education*, 160, 104023. [CrossRef]
- Yamashita, K., Fujioka, R., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2016). Practices of algorithm education based on discovery learning using a program visualization system. *Research and Practice in Technology Enhanced Learning*, 11(1), 15. [CrossRef] [PubMed]
- Yusuf, A., & Noor, N. M. (2023). Research trends on learning computer programming with program animation: A systematic mapping study. *Computer Applications in Engineering Education*, 31(6), 1552–1582. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.