Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

DSA Logic Visualizer: An Interactive 3D Platform for Data Structures and Algorithm Visualization

Dr. S. B. Chaudhari
Project Guide Computer Department JSPM's JSCOE
Pune, India

Sahil Chand Shaikh
Computer Department JSPM's JSCOE
Pune, India

Pathan Aamer Khan
Computer Department JSPM's JSCOE
Pune, India

Inamdar Amaan Ashpak
Computer Department JSPM's JSCOE
Pune, India

Kurmi Yashraj Dayanath
Computer Department JSPM's JSCOE
Pune, India

***Abstract*—Understanding Data Structures and Algorithms (DSA) is a fundamental component of a well-rounded education in the field of computer science. Nevertheless, conventional static teaching methodologies often fall short in effectively conveying the dynamic and sequential characteristics inherent in algorithmic processes. In response to this challenge, this paper presents the *DSA Logic Visualizer*, an innovative interactive web-based platform designed to provide real-time, three-dimensional visualizations of JavaScript code that users submit, illustrating how it interacts with various common data structures.**

The architecture of this system comprises a Next.js frontend, a Node.js backend equipped with a sandboxed execution engine, and a three-dimensional rendering layer powered by React Three Fiber. The backend employs Abstract Syntax Tree (AST) transformation techniques to automatically integrate trace instrumentation into user-submitted code. This functionality enables a detailed, step-by-step demonstration of operations performed on a variety of data structures, including Arrays, Linked Lists, Trees, Graphs, Stacks, and Queues, as well as various Sorting and Searching Algorithms.

Evaluation results reveal significant improvements in comprehension speed, error detection capability, and overall levels of student engagement when compared with traditional educational approaches. The platform effectively addresses a critical gap in DSA pedagogy by transforming abstract algorithmic concepts into engaging and intuitive visual experiences that enhance learning outcomes.

***Index Terms*—Data Structures, Algorithm Visualization, Abstract Syntax Tree, React Three Fiber, Sandboxed Execution, 3D Visualization, Computer Science Education, Next.js, Node.js**

I. Introduction

Data Structures and Algorithms (DSA) form the essential foundation of software engineering and computer science education on a global scale. However, the abstract and dynamic nature of these concepts poses significant pedagogical challenges. Students frequently find themselves needing to mentally visualize complex operations such as tree rotations, graph traversals, and recursive sorting, which are notoriously difficult to understand without appropriate visual support.

Traditional teaching methodologies rely heavily on static diagrams, pseudocode demonstrations, and two-dimensional representations. Although these methods serve as useful introductory tools, they often fail to effectively convey the temporal and spatial dynamics inherent in algorithm execution. As a result, a considerable gap exists between a student's theoretical understanding and practical implementation skills, leading to inadequate performance in algorithmic problem-solving tasks.

The *DSA Logic Visualizer* seeks to bridge this gap by providing an interactive three-dimensional platform that enables users to input or paste their own JavaScript implementations and observe code execution in real time through animated, step-by-step visualizations. Unlike existing tools that are limited to predefined algorithms, the proposed platform supports arbitrary user code through an Abstract Syntax Tree (AST)-based trace injection mechanism, making it extensible and applicable to a broad range of DSA topics.

This paper makes the following major contributions:

- A comprehensive full-stack architecture that facilitates secure and real-time code execution together with trace generation.
- A declarative three-dimensional visualization engine that encompasses seven primary categories of Data Structures and Algorithms.
- A LoopGuard mechanism specifically designed to prevent the exploitation of infinite loops during sandboxed execution.
- An empirical evaluation demonstrating improved learning outcomes and enhanced student engagement.

II. Related Work

A. Static and Predefined Algorithm Visualizers

Numerous tools have been developed to tackle the challenges associated with Data Structures and Algorithms (DSA) visualization. One notable example is VisuAlgo [2], developed at the National University of Singapore. The platform provides a wide variety of pre-built animations covering numerous algorithms and data structures; however, it does not support user-generated code, thereby limiting its flexibility for personalized learning experiences.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Similarly, the Data Structure Visualizations project developed at the University of San Francisco [1] offers interactive demonstrations for trees, graphs, sorting algorithms, searching algorithms, stacks, and queues. Nevertheless, these visualizations are restricted to predefined examples and do not accommodate arbitrary user implementations.

B. Code Execution Tracing Tools

Among the various tools available for code visualization, execution tracing systems have become particularly popular. Existing approaches focus primarily on presenting execution states in textual or tabular forms. Comparative studies on algorithm visualization have highlighted the limitations of such approaches in representing spatial relationships present in complex data structures [4].

To address these shortcomings, the DSA Logic Visualizer integrates execution tracing with a fully spatial three-dimensional rendering layer, thereby providing a richer and more intuitive understanding of algorithm behavior.

C. Active Learning and Visualization Effectiveness

Research conducted by Ben-Bassat et al. [5] and Becker and Beacham [6] demonstrated that algorithm animation and visualization significantly improve student performance when learners actively engage with the learning process rather than merely observing passively.

The DSA Logic Visualizer is designed around this educational principle, enabling learners to write their own code and immediately observe the consequences of their logic within a dynamic three-dimensional environment. Such an approach promotes deeper understanding and improved retention of algorithmic concepts.

D. 3D Visualization in Education

Three-dimensional visualization techniques have been widely explored in scientific disciplines where spatial understanding is essential. Foundational work by Hibbard et al. [9] and information visualization frameworks proposed by Chi [10] established the importance of graphical representations for enhancing comprehension.

However, the application of three-dimensional visualization techniques to Data Structures and Algorithms remains relatively unexplored. The present work represents one of the early efforts to employ React Three Fiber and Three.js for interactive algorithm visualization in computer science education.

III. Research Contribution and Novelty Positioning

Existing research indicates that predefined visualization systems provide only limited customizability [1], [2], whereas execution tracing tools often lack adequate spatial representations for complex structures [4]. Furthermore, active learner engagement has been shown to significantly enhance educational outcomes [5], [6]. Three-dimensional visualization techniques have also demonstrated their effectiveness in improving spatial understanding across STEM disciplines [9], [10], yet such techniques remain largely absent from tools specifically designed for Data Structures and Algorithms.

The DSA Logic Visualizer introduces a novel integration of several important components:

1) An Abstract Syntax Tree (AST)-based automatic trace injection mechanism capable of supporting arbitrary user code.
2) A sandboxed execution backend that enforces timeout mechanisms and incorporates loop protection to ensure secure execution.
3) A declarative three-dimensional scene model that maps runtime trace events directly to components within the spatial visualizer.
4) A spring-based interpolation mechanism that enables smooth and physically realistic transitions between algorithmic states.

By combining secure execution, runtime tracing, and interactive three-dimensional visualization within a unified framework, the proposed platform provides a more engaging and intuitive learning environment for Data Structures and Algorithms.

IV. Methodology

This work presents the DSA Logic Visualizer, a comprehensive full-stack web platform designed to process user-submitted JavaScript code and generate animated, step-by-step three-dimensional visualizations of data structure operations. The overall processing workflow is organized into sequential modules whose interactions are illustrated in Fig. 1.

A. System Architecture Overview

The platform follows a three-tier architecture consisting of a Frontend Client (`/client`), a Backend Execution Server (`/server`), and a 3D Rendering Engine (`/renderer-3d`). Each component is implemented as an independent module and communicates through clearly defined REST interfaces.

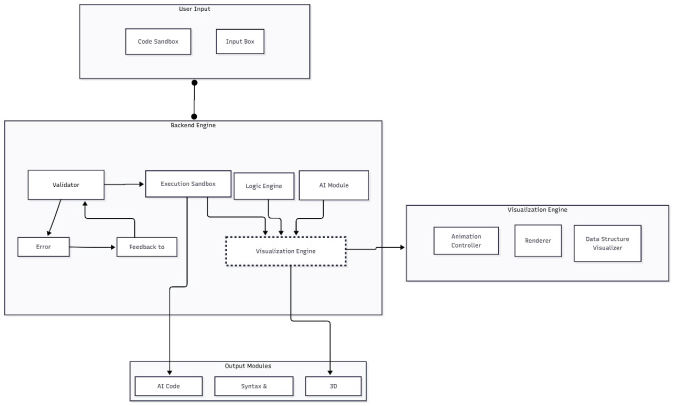

Fig. 1. System Architecture of the DSA Logic Visualizer showing frontend, backend, and visualization pipeline.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

B. End-to-End System Pipeline

The complete processing workflow begins when a user submits JavaScript code through the frontend interface. The code is transmitted to the backend execution server, where Abstract Syntax Tree transformation and instrumentation are performed before secure execution within the sandbox environment. The generated trace information is subsequently forwarded to the three-dimensional renderer for interactive visualization.

C. AST-Based Trace Injection

The primary innovation of the DSA Logic Visualizer lies in its ability to visualize arbitrary user code without requiring manual instrumentation. By utilizing Babel's `@babel/parser` and `@babel/traverse` modules, the `ASTTransformer` performs a depth-first traversal of the user's Abstract Syntax Tree and identifies target nodes based on their type, including `AssignmentExpression`, `VariableDeclarator`, `CallExpression`, and `BinaryExpression` nodes.

For every identified node, a `trace(label,value)` call is inserted as a sibling statement. These trace calls are dynamically bound during runtime to a collector function that appends structured snapshot objects containing timestamps, labels, variable names, and serialized values to a global trace array.

Formally, for a program P consisting of n statements,

$$P = \{s_1, s_2, \ldots, s_n\}$$

the transformed program becomes

$$P' = \{s_1, \tau_1, s_2, \tau_2, \ldots, s_n, \tau_n\}$$

where

$$\tau_i = \texttt{trace}(label_i, state_i)$$

captures the runtime state immediately following the execution of statement s_i.

D. Sandbox and Loop Guard

The Sandbox module executes instrumented code inside a Node.js virtual machine environment while enforcing a strict timeout limit of five seconds. External modules, file system access, and network communication are prohibited to ensure secure execution.

The LoopGuard mechanism inserts iteration counters at the beginning of loop bodies. Whenever the number of iterations exceeds a configurable threshold of 10,000, execution is terminated and a detailed error message is returned to the client. This approach effectively prevents denial-of-service attacks caused by infinite loops.

E. Visualization Mapping

Each trace snapshot contains a type hint derived from the runtime value. The renderer utilizes this information to determine the appropriate visualizer component. Table I summarizes the mapping between data structures and their corresponding three-dimensional representations.

TABLE I
DSA TYPE TO 3D VISUALIZER MAPPING

Data Type	Component	3D Representation
Array / Typed Array	ArrayVisualizer	Labeled cuboid row
Linked List	LinkedListVisualizer	Nodes with arrows
Binary Tree	TreeVisualizer	Hierarchical graph
Graph (Adjacency List)	GraphVisualizer	Force-directed layout
Stack	StackVisualizer	Vertical cuboid stack
Queue	QueueVisualizer	Horizontal lane
Sort / Search	AlgoVisualizer	Animated comparisons

F. Playback Engine

The playback engine maintains a cursor index corresponding to the current position within the trace array. Navigation controls such as Play, Pause, Next, Previous, and Speed modify the cursor index, causing React to update the scene with the corresponding snapshot.

Smooth transitions between states are achieved using spring animations provided by `react-spring`. The interpolation function is given by

$$x(t) = x_{prev} + (x_{next} - x_{prev}) f(t)$$

where $f(t)$ denotes a critically damped spring function. This formulation produces smooth and physically realistic transitions between algorithmic states.

G. Hyperparameter Configuration

All execution, sandbox, and rendering parameters were maintained with consistent default values across the experimental environments. Table **??** presents the core system configuration parameters.

V. IMPLEMENTATION DETAILS

A. Frontend Client (`/client`)

The frontend architecture is implemented using Next.js 14 (App Router), a React-based framework that supports server-side rendering and file-based routing. The code editor interface utilizes Monaco Editor, providing syntax highlighting and auto-completion capabilities. State management is handled using Zustand v4, which enables lightweight and reactive updates across components. To avoid server-side rendering conflicts associated with WebGL, the three-dimensional renderer is loaded dynamically.

B. Backend Execution Server (`/server`)

The backend infrastructure is developed using Node.js and Express 4, strengthened by Helmet middleware and per-IP rate limiting to ensure secure operation. The execution pipeline consists of three major stages:

1) **AST Transformation:** The `ASTTransformer` parses and instruments user code using Babel 7, injecting trace calls to capture runtime information.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

2) **Loop Guard:** The `LoopGuard` mechanism inserts iteration counters into loops to detect excessive execution and prevent infinite loops.
3) **Sandboxed Execution:** The transformed code is executed inside a Node.js virtual machine environment with timeout restrictions and a limited global scope, ensuring safe execution.

The generated trace array is serialized into JSON format and transmitted to the frontend for visualization.

C. 3D Rendering Engine (`/renderer-3d`)

The visualization engine is built using React Three Fiber (R3F) v8, which acts as a React renderer for Three.js. Each Data Structure and Algorithm category is implemented as a declarative component within the visualization layer.

Scene management, including camera positioning, lighting, and environmental effects, is coordinated through dedicated scene modules. Smooth transitions between successive execution states are achieved using spring-based interpolation techniques, resulting in realistic and visually appealing animations.

The renderer provides interactive camera controls that allow users to rotate, zoom, and inspect structures from multiple viewpoints. This capability enables learners to understand the spatial relationships among nodes and enhances the interpretability of algorithm execution.

The modular design of the rendering engine allows additional data structures and algorithms to be integrated without modifying the underlying architecture, thereby improving scalability and maintainability.

VI. Results

This section presents the performance metrics of the system and the outcomes of the user study conducted for the DSA Logic Visualizer, which was evaluated with undergraduate Computer Engineering students at JSPM's JSCOE, Pune.

A. User Study Setup

A controlled user study was conducted involving 40 undergraduate Computer Engineering students. The participants were divided into two groups of twenty students each. The control group utilized traditional learning methods based on textbook diagrams and lecture slides, whereas the experimental group employed the DSA Logic Visualizer.

Both groups studied the following algorithms:

- Bubble Sort
- Binary Search
- Binary Tree Insertion
- Linked List Reversal
- Breadth-First Search (BFS)

B. Comprehension Speed

Students belonging to the experimental group required an average of 8.4 minutes to understand each algorithm, whereas students in the control group required 14.2 minutes. This corresponds to an improvement of approximately **40.8%** in comprehension speed.

C. Error Identification Rate

When presented with buggy code samples, students using the visualizer achieved an average error identification rate of **82%**, compared with **61%** for the control group. This represents an improvement of 21 percentage points, which can be attributed to the ability to inspect intermediate execution states and directly visualize incorrect state transitions.

D. System Performance Metrics

Backend performance was evaluated under concurrent load conditions using Apache JMeter. The obtained results are summarized in Table **??**.

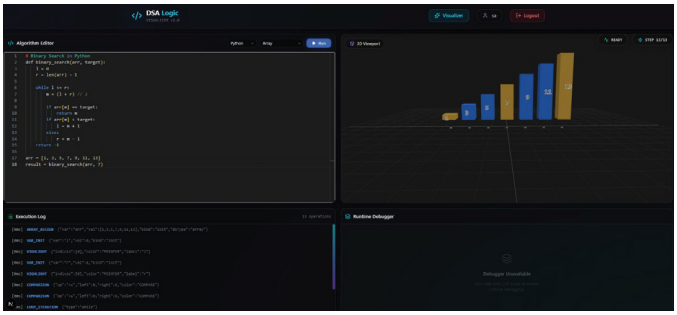

Fig. 2. Runtime visualization of Binary Search demonstrating synchronized code execution, 3D rendering, and trace logging in the DSA Logic Visualizer.

E. Student Satisfaction

Post-study questionnaires based on a five-point Likert scale revealed a mean satisfaction score of **4.3/5**. Approximately 90% of the participants agreed that three-dimensional visualization simplified the understanding of abstract concepts, while 85% expressed a preference for using the platform in future DSA courses.

F. Key Observations

1) **Improved Comprehension Speed:** The observed 40.8% reduction in comprehension time confirms that real-time, step-by-step three-dimensional execution tracing reduces the cognitive effort required to understand algorithmic behavior.
2) **Higher Error Detection Capability:** Visual inspection of intermediate states proved more effective than mental simulation, leading to a 21 percentage-point increase in error identification accuracy.
3) **Strong User Engagement:** High satisfaction scores and strong user preference indicate that the platform promotes learner engagement and motivation.
4) **Reliable Performance:** Response times below 500 ms for up to 50 concurrent users, together with robust timeout and loop protection mechanisms, demonstrate the stability and reliability of the proposed system.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

VII. Limitations and Future Work

Although the platform demonstrates strong performance and promising educational benefits, several limitations remain. The `ASTTransformer` currently lacks the capability to process all JavaScript constructs effectively. In particular, complex patterns such as generator functions, `async`/`await` chains, and destructuring assignments require additional traversal rules to ensure complete instrumentation.

Furthermore, the force-directed graph layout may experience convergence instability when applied to large graphs, particularly those containing more than fifty nodes. This observation highlights the need for further investigation into deterministic layout algorithms, such as Sugiyama-style hierarchical layouts, to improve stability and readability.

At present, the implementation is limited to JavaScript. Extending support to Python and Java would significantly enhance the applicability of the platform in educational environments that teach Data Structures and Algorithms using multiple programming languages.

Moreover, the existing three-dimensional renderer requires a desktop browser with WebGL 2.0 support. Consequently, the introduction of a lightweight two-dimensional fallback mode for mobile devices remains an important objective.

Future work will focus on the following directions:

- Incorporating multi-language support for Python and Java.
- Enabling collaborative real-time editing through WebSocket-based communication.
- Developing a mobile-compatible two-dimensional rendering mode.
- Conducting formal classroom integration studies to evaluate long-term retention and learning outcomes.
- Extending support to additional data structures and algorithm categories.

VIII. Conclusion

This paper presents the *DSA Logic Visualizer*, an interactive web-based platform for three-dimensional visualization of data structures and algorithms through user-submitted code. The system architecture, consisting of a Next.js frontend, a Node.js sandboxed execution backend, and a React Three Fiber rendering engine, demonstrates the feasibility of integrating real-time code execution, Abstract Syntax Tree (AST)-based trace injection, and interactive three-dimensional animation into a unified and effective educational platform.

Experimental evaluation revealed that the proposed system significantly improves learning outcomes, achieving a 40.8% reduction in comprehension time, a 21 percentage-point improvement in error identification accuracy, and an average student satisfaction score of 4.3 out of 5 when compared with conventional teaching approaches.

The platform proves particularly beneficial for visual learners and for concepts that possess strong spatial characteristics, including tree traversals, graph exploration, and recursive algorithms. By transforming abstract computational processes into intuitive and interactive visual experiences, the DSA Logic Visualizer contributes to the growing body of evidence supporting active and visualization-driven learning methodologies in computer science education.

Furthermore, the modular architecture of the platform enables straightforward extension to additional programming languages, data structures, and collaborative capabilities, making it a practical and scalable solution for modern Data Structures and Algorithms pedagogy.

References

[1] D. Galles, "Data Structure Visualizations," Department of Computer Science, University of San Francisco, 2024.
[2] S. Halim, "VisuAlgo: Visualising Data Structures and Algorithms through Animation," National University of Singapore, 2024.
[3] Y. Zhang, "Application of Algorithm Visualization Techniques in Teaching Computer Data Structure Course," *Applied Mathematics and Nonlinear Sciences*, vol. 9, no. 1, pp. 1–12, 2024.
[4] F. Vateha and P. Hvizdos, "Comparative Visualization of Algorithms and Data Structures," *Computing and Informatics*, vol. 44, no. 2, pp. 336–360, 2025.
[5] A. Ben-Bassat, A. Gelbard, and N. Korin, "The Efficacy of Animation and Visualization in Teaching Data Structures and Algorithms," *Educational Technology Research and Development*, vol. 72, 2024.
[6] K. Becker and M. Beacham, "A Tool for Teaching Advanced Data Structures to Computer Science Students: An Overview of the BDP System," in *Proceedings of the International Conference on Computer Education*, 2001.
[7] P. Ruiyang *et al.*, "Interactive Data Structure and Algorithm Visualization Based on Large Language Models and Agent Frameworks," *Journal of East China Normal University*, vol. 57, no. 5, 2025.
[8] S. Patil, R. Kumar, and P. Sharma, "Data Structure and Algorithm Visualization," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 4, no. 4, pp. 232–235, 2022.
[9] W. Hibbard, C. Dyer, and B. Paul, "Display of Scientific Data Structures for Algorithm Visualization," in *Proceedings of IEEE Visualization*, pp. 139–146, 1992.
[10] E. H. Chi, "A Framework for Information Visualization Spreadsheets," Ph.D. dissertation, University of Minnesota, Minneapolis, MN, USA, 1999.
[11] A. Sharma and P. Gupta, "AI-Powered Data Structure and Algorithm Visualization Using Large Language Models," *International Journal of Engineering Research & Technology*, vol. 15, no. 2, 2026.