Web-based Visualization Tools to Demonstrate the Working of Sorting and Pathfinding Algorithms

Shankar M. Patil
Department of Information Technology
Bharati Vidyapeeth College of Engineering, Kharghar
Navi Mumbai, India

Ankit Joshi
Department of Information Technology
Bharati Vidyapeeth College of Engineering, Kharghar
Navi Mumbai, India

Nikhil Sawant
Departmentof Information Technology
Bharati Vidyapeeth College of Engineering, Kharghar
Navi Mumbai, India

Atharva Jagdale
Department of Information Technology
Bharati Vidyapeeth College of Engineering, Kharghar
Navi Mumbai, India

ABSTRACT

This paper focuses on introducing web-based visualization tools that demonstrates the working of famous sorting and pathfinding algorithms. These tools aim toward providing abstract thinking to different algorithms by making use of animations and 3D models which will stimulate the working of each separate algorithm in a unique way. To demonstrate the working of path-finding algorithms, 3D visualization tool having user-defined maze will be provided where each algorithm will be animated in a unique way to show how it traverses to find the shortest path between source and destination This maze will be constructed using WebGL libraries like react-three-fiber and external models like building representing path blocks will be loaded using GLTF (Graphics Language Transmission Format)Loaderprovided by ThreeJS. The second tool focuses on sorting algorithms which make use of each unique iteration of CSAlgorithms like bubble sort and animate its working by making use of a bar chart which will represent the array to be sorted. After selecting a sorting algorithm, these bars will be compared, swapped and animated to show working of these algorithms. Both the tools will be highly customizable and user input driven to provide them with a powerful and interesting tool to learn and experiment with different CSAlgorithms. Both the tools will be rendered in React DOM and work on principle that once an algorithm is executed its iteration and flow logs are captured which are used for performing animation using JavaScript and TweenJS libraries.

Keywords

Data Structure, Pathfinding Algorithm , Sorting Algorithms , Visualization, Computer Science Algorithms,Animation, 3D Models.

1. INTRODUCTION

Data structures and Path Finding algorithms are of paramount importance in computer algorithms. Complete understanding of all these concepts is of the utmost importance for students which helps in developing problem-solving skills. It is important for IT Developers to solve problems with the help of programming languages, and data structures these are keys to solving problems. Algorithms are the state of the art of computer science. There are many ways to solve the same problem, and some methods are better than others. Using the right algorithm in the right place can save a lot of time and money.
For example, there are at least three or four ways to find the shortest path to reach the destination, but knowing that the A-star algorithm is more efficient than the Dijkstra’s algorithm, will lead to a much more efficient path-finding process, also it’s fairly easy to implement a sequential or linear search algorithm for a list of data, but knowing that the binary search algorithm can sometimes be twice as efficient as the sequential search will lead to a better program.
Thus, by understanding the importance of data structures and pathfinding algorithms they have become an important part of computer science and technical recruitment. There are different learning methods like books, presentations, and online classes which are used by students, teachers, and developers to learn these algorithms. However, understanding the complete working of these algorithms requires conceptual thinking. To solve these problems recently there are various new 2D virtual tools emerging which explain the working of these algorithms step by step. However, understanding the working of a complex algorithm like graph traversals 2D faces some limitations. Taking advantage of advancements in WebGL, it can be used to upgrade these tools to 3D to give users in-depth working knowledge of these algorithms.

2. RESEARCH METHODS

There are three stages in conducting this study. In the first phase, a literature review was conducted to find a model based on existing related activities and theories. The basic structures and their relationships needed to be first perceived and understood before it could be used as a conceptual model for tool design.
The second phase was designing the technologies that were to be used, working on them to build the tool and discussing its scope. The identified constructs had to be supported by the tool, so later how they affected learning outcomes were possible to be examined. Therefore, the tool had to be designed and developed carefully so that it could support the presence of all the identified constructs.
In the third phase, the tool was tested on learners to measure to what extent the tool, as the representation of cognitive support and engagement, influences learners’ outcomes.

3. RELATED WORKS

Over the years, there have been numerous studies and papers on the use of filtering techniques as visual aids. One is a complete overview of how to do animation and do mathematical analysis, while others focus on different techniques aimed at further understanding of the same animation. Several methods have been developed to deliver algorithms in a tangible way that students can easily understand. Visualization is a common approach to planning and programming, a task that translates algorithms into applications, taught using the following methods: traditional expressions and labs, robots, problem-based learning, software visibility, and programming areas and support tools. The learning-teaching process is still very tedious for teachers, ranging from fine-tuning to multidisciplinary testing among publications including support tools, 78.6% of CS teaching-related techniques [3]. In 2011, Dr. Steven Halim conceptualized a tool called visual algo which helps students understand data structures and algorithms in an efficient manner. This tool provides an interactive playground for the users where they can give the input data structure, apply constraints and visualize algorithm animation. While animation of algorithms, the end user is also able to see the pseudo code with steps which are executed at that particular time which helps them to understand the algorithm in an efficient way.

Research in Virtual reality technology has found that WebGL libraries can be used to develop 3D scenes and develop web guide systems [5]. Performance tests show that the much-needed loading data for animation improves the quality of user interaction information by reducing the amount of data previously loaded. However, the number of animated tracks and the size of the private animation data affect the setting of this method [1]. There is a method based on 3DS MAX technology to optimize the model memory. The modeling methods are based on the basic modeling library, composite modeling library, surface toads modeling library, polygon modeling library, facets modeling library, NURBS modeling library more than six types of model library and it identifies if the model is duplicated. Although, the research on model texture mapping technology still needs to be improved. It should focus on the optimization of texture mapping [10].

Alternative to ThreeJS is provided by unity WebGL support which allows developers to develop 3D scenes with a very powerful game engine and simply make its build for the web. Pathfinding algorithms can be used in unity for different games and research applications [6].

There is research going on improving performance of 3D scenes by simplification of the scene by removing objects that are not currently visible or degrading the complexity of the models from the distance of the observer and creating an image from 2D and 3D objects. Consequently, the computer graphics comes in with the employment of interface Open Graphics Library (OpenGL) for Android and new Metal API (Application Programming Interface) introduced with IOS8. But the only issue here is level of detail (LOD) does not solve a problem that is directly visible in the graphic chain but it is perceptible in the complex scene [11].

The main advantage of rendered 3D maps is that some of the maps like Mapbox use a few simplifications for speeding up rendering. For example, when it is not possible to get the height information of a building, but it can be estimated to be 4 stories tall, developers can tag those width buildings: levelsis equal to four in OpenStreetMap Mapbox Streets processing code will convert that to 12 meters in the vector tiles since it approximates the height of one level as 3 meters (about 10 feet) [2]. The optimization efficiency is not significant for a 3D model that contains only one animation track, and the animation data volume is small (e.g., less than 50 kb) if Java Applets are considered using Swings, and AWT [4]. Direct Canvas mode has a great impact on performance and memory. Indeed, it can be used to play WebGL games on mobile and the TV smoothly. Also, the amount of 3D memory decreased with Direct Canvas mode. That is because this mode does not create intermediate buffers with the window size anymore. But the only drawback is that there are already optimized WebGL renderers present like ThreeJs (a JavaScript library) which are specifically made for websites [7]. WebGL programs consist of control code written in JavaScript and shader code that is executed on a computer's Graphics Processing Unit (GPU). As it is too complicated programming with WebGL directly, the framework three.js has been added to the system. Three.js allows the creation of GPU accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins [8]. The logical structure and physical storage format of the 3D spatial data, & can effectively divide and index the 3D scene. Results show that the proposed data structure can save about 60%-80% disk space, and the improved index structure is more sufficient [9].

4. DESIGN CONSIDERATIONS AND IMPLEMENTATION

The main purpose of the tool is to provide users with easy to use and interactive playgrounds to visualize different algorithms and learn at their own pace. It's important to use a well-defined tech stack to make these tools to achieve all the functionalities easily and efficiently.

4.1 Frontend

According to Statista, react is the most used framework for the year 2021 where it states almost 40.14% of reported software developers were using react. React is a JavaScript web framework which allows developers to create reusable models and have higher performance websites.

4.2 Backend and external tools

For the backend, NodeJS is the ideal option for the website to provide high performing API's. NodeJS is JavaScript runtime framework which has gained huge popularity and is supported by many NPM libraries which provides great utility for developers. Blender is an open source and free 3D computer graphics software which can help to make 3D models for the visualization tool. These models can later be easily integrated using react-three-fiber library in react.

4.3 User Interaction with the tool

Fig 1 shows how users interact with sorting and pathfinding visualization tools. When interacting with visualization tools, First users have to provide inputs like array, algorithm, speed of animation and in case with 3D pathfinding visualization tool, constructing the maze in 3D scene. Once inputs and configurations are set, user can trigger visualization which will start visualization of selected algorithm

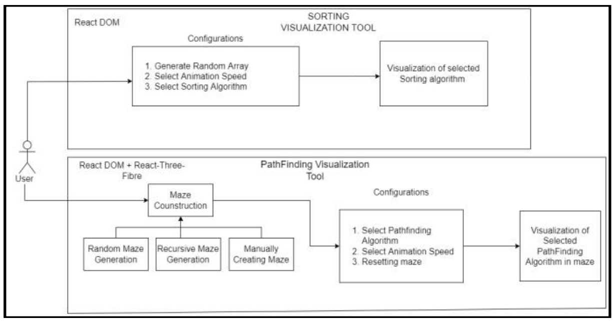
React DOM
SORTING VISUALIZATION TOOL
Configurations
1. Generate Random Array
2. Select Animation Speed
3. Select Sorting Algorithm
Visualization of selected Sorting algorithm
User
React DOM + React-Three-Fibre
PathFinding Visualization Tool
Maze Counstruction
Random Maze Generation
Recursive Maze Generation
Manually Creating Maze
Configurations
1. Select Pathfinding Algorithm
2. Select Animation Speed
3. Resetting maze
Visualization of Selected PathFinding Algorithm in maze

Fig. 1: A block diagram showing interaction of user with visualization tool

4.4 Implementation of the Sorting Algorithms in visualization tool

All the algorithms discussed are visualized in web applications by using bars. Each bar having height

corresponding to its value represents an element of the array. Once the animation starts, bars are swapped according to algorithm iteration thus visualizing the working of the algorithm. The bars which are in comparison will blink and change into different colors. If the numbers in comparison satisfy the condition, then accordingly the position of those bars will be swapped. The animation speed of the tools can further be adjusted according to the user. Once the visualization of the algorithm finishes each iteration of the algorithm is also printed.
As shown in fig 2 depicts the general flow of how animation is produced for different sorting algorithms. First the user gives the randomly generated array to the sorting algorithm. The aim of the sorting algorithm is to efficiently sort the array and create the animation array which contains actions performed by algorithms while sorting the array. This log is an array which stores information like comparisons indexes and swap indexes. Later these animated arrays having logs are given input to the animation engine which iterates through logs and presents required animation for that log.

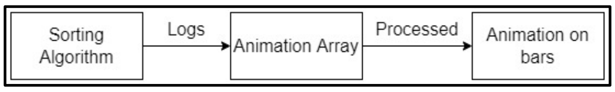

Fig. 2. Internal working of sorting visualization tool

Here are steps or pseudo code to generate this action array for different algorithms. This action array is referred to as the animation array below. Also with animation, an iteration array is also generated for showing iterations to the end user.

A. Bubble Sort Algorithm for generating animation array:

These are lists of variables used in steps.

1. auxillaryArray = Input array to be sorted
2. animations = algorithm action array to be used for animation
3. iterationArray = Array containing iterations performed by algorithm

```
Step 1: Define N = auxillaryArray.length, dummy[] and iter=N-1
Step 2: While (iters> 0)
         Set swapped = false
Step 3: For i = 0 to iters
           animations.push(["comparision1", i, i + 1])
           animations.push(["comparision2", i, i + 1]
Step 4: If  auxillaryArray[i] >auxillaryArray[i + 1]
         Set swapped = true
Step 5: animations.push(["swap", i, auxillaryArray[i + 1]])
        animations.push(["swap", i + 1, auxillaryArray[i]])
        swap(auxillaryArray, i, i + 1)
Step 6: dummy=auxillaryArray.slice()
    iterationArray.push(dummy)
        if (swapped === false) break
        iters--
Step 7: return {animations , iteratonArray}
```

B. Merge Sort Algorithm for generating animation array:

These are list of variables used in steps

1. auxillaryArray = Input array to be sorted
2. animations = algorithm action array to be used for animation
3. startIndex= first element index of the array
4. endIndex = last element index of the array

Steps for merge sort generating animation array

```
Step 1: if startIndex === endIndex
      return middleIndex = Math.floor((startIndex +
          endIndex)/2)
Step 2: Recursive Call function mergeSort(auxillaryArray, startIndex, middleIndex, animations);
Step 3: Recursive Call function mergeSort(auxillaryArray, middleIndex + 1, endIndex, animations)
Step 4: Call function merge(auxillaryArray, startIndex, middleIndex, endIndex, animations)
```

Steps for merge sort helper function

```
Step 1: Define sortArray = [], i = startIndex, j = middleIndex + 1
Step 2: While(i <= middleIndex&& j <= endIndex)
    animations.push(["comparision1", i, j])
    animations.push(["comparision2", i, j])
Step 3: if auxillaryArray[i] <= auxillaryArray[j]
      sortArray.push(auxillaryArray[i++])
    else
      sortArray.push(auxillaryArray[j++])
Step 4: while(i <= middleIndex)
    animations.push(["comparision1", i, i])
    animations.push(["comparision2", i, i])
    sortArray.push(auxillaryArray[i++])
Step 5: while(j <= endIndex)
    animations.push(["comparision1", j, j])
    animations.push(["comparision2", j, j])
    sortArray.push(auxillaryArray[j++])
Step 6: For i = startIndex to endIndex
    animations.push(["comparision1", i, i - startIndex])
   animations.push(["overwrite", i, sortArray[i - startIndex]])
    animations.push(["comparision2", i, i - startIndex])
    auxillaryArray[i] = sortArray[i - startIndex]
Step 7: End
```

Animation action array generated in the above algorithm is a 2D array containing swapping information in each iteration the schema of the array contains the names of comparisons and comparing indexes as shown in fig 3.

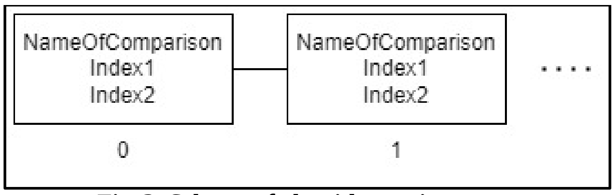

Fig. 3: Schema of algorithm action array

This array is processed by a custom animator function which iterates through these arrays in a timeout interval to get animation instruction, For example, based on comparison name two indexes are given the same color as shown in fig 4 or name of comparison can also suggest that swapping is required. In this way a unique animation cycle is generated for each sorting algorithm giving the user the best experience for learning.

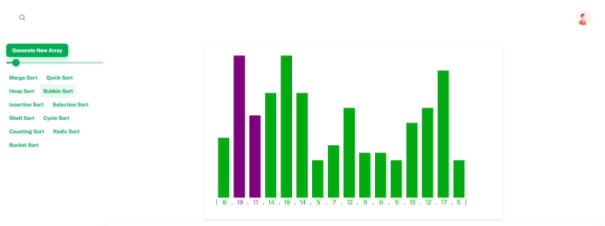

Fig. 4: Animation to show how two elements in array are compared by giving different color

4.5 Implementation of the 3D grid using ThreeJS and react-three-fiber

Instead of using divs and spans, the react-three-fiber (R3F) helps to supply three objects.js such as matches, lights, cameras, and shades. It provides 3D items to the requirements of users in the marketplace with the help of Three.js key code under the lid and in the same way that react provides access to the DOM archives using ref, the react-three-fiber allows access to ThreeJS artifacts if control is needed. The easy configuration of the scene camera allows an immersive 3D experience for users.

In order to visualize path-finding algorithms, a 3D grid is used to represent different square nodes. This grid is rendered using GridHelper provided by react three fiber and later geometry and material is applied to these grid Geometry is a representation of mesh, line, or point geometry. It includes vertex positions, face indices, normal, colors, UVs, and custom attributes within buffers, reducing the cost of passing all this data to the GPU. Grid geometry is determined by the geometric category of the plane provided by ThreeJS. It takes different parameters like width, length, sections of width, half-length. Material defines the appearance of an object. It includes color, combination, etc.

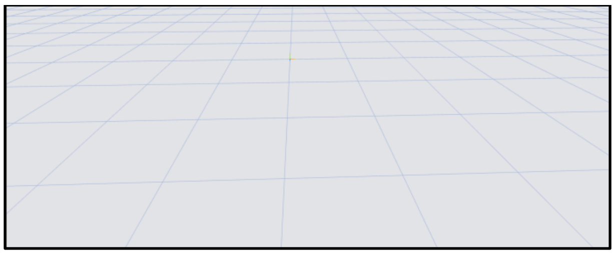

Fig. 5: Grid rendered using GridHelper provided by react-three-fiber library

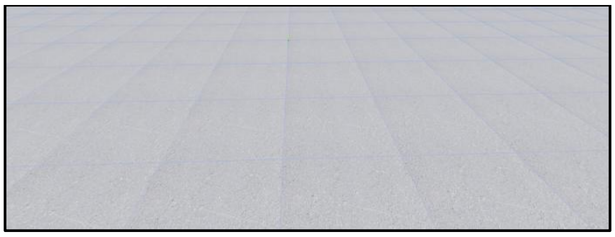

Fig. 6: Grid after applying geometry and material

The nodes of the grid can represent either start position , destination position , path or walls. Start and end position can be predefined or user input driven. By default, except start and destination nodes are path nodes. Path blocks are represented as buildings which are rendered using gltf loader. This external 3D model of the building is spawned once the user clicks on the node. For faster loading of similar .glb models copy of its geometry is done after its first load which is used to render the next building.

Grid can be constructed in three ways, manually clicking on the grid node, randomly generating maze and recursive maze generation. Random wall is generated using the Math, Random function. Each node in the grid can be accounted as a 2D array. This 2D array is iterated and is assigned with a wall or path. Here in figure 7 the building represents the path block.

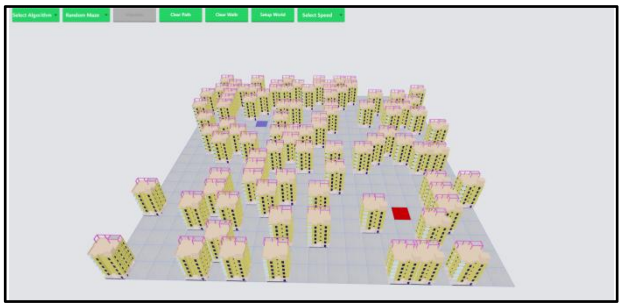

Fig. 7: Result taken after a random maze was generated

The third way to generate a maze is recursive strategy. The depth-first search algorithm of maze generation is frequently implemented using backtracking. This can be described with the following recursive routine**.** Here is an algorithm for recursive strategy.

1. Given a current cell as a parameter.
2. Mark the current cell as visited.
3. While the current cell has any unvisited neighbor cells.
   - Choose one of the unvisited neighbors.
   - The wall between the current cell and the chosen cell will be removed.
   - Invoke the routine recursively for a chosen cell which is invoked once for any initial cell in the area.

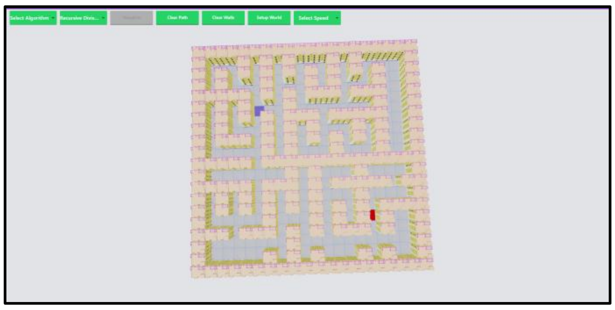

Fig. 8: Maze generated using recursive strategy

Next step is to visualize the algorithm in a 3D maze. Grid nodes are mapped with terrain array containing nodes information like position, value and 3D models configurations. This array will be kept in synchronization with another 2D array which will be processed by algorithms to process and generate animation instructions. The animation is divided into two parts. First is animating the nodes traversal performed by algorithm and second is for animating the shortest path from start to end position. Let's take an example of how the Dijkstra algorithm will be animated in this maze.

Algorithm to generate animation array:

Step 1: Select the source node as the initial node and other nodes to have infinite path
Step 2: Collect all the unvisited nodes.
Step 3: Iterate through these unvisited nodes

Step 4: In this iteration the first step is to select the closest node among all neighbors connected nodes and ensure it is not a path block.
Step 5: Check distance of selected closest node if it is infinity, return false to stop loop.
Step 6: If the distance is not infinity, Push closest node is in animation array
Step 7: Update neighbors' nodes distance. If it is target node loop is stopped
Step 8: Iterate Step 4 to 6 until loop is break or all unvisited nodes are visited

The animation array generated here will be used to do node traversal animation using TweenJS library on the 3D grid. Next part is to generate a backtracking array containing nodes of the final shortest path.

Algorithm to generate animation array for backtracking final path

Step 1: mark the current node as previous of finish node to exclude it from animation
Step 2: Push the current node in the animation array at the end.
Step 3: Point the current node to its previous node
Step 4: Repeat step 2 and 3 until previous of current node is not start node

This backtracking array of nodes will be used for animating the final path.

5. RESULTS AND DISCUSSIONS

5.1 Interaction with sorting visualizer tool

The goal of the sorting visualizer tool is to make use of bars and their comparisons to help understand sorting algorithms. This is user flow which shows how sorting visualizer tools can be used.
Step 1: User clicks on Random Array Generator Button to generate a random array which will be used for visualization. These generated a random array which is represented by a bar.
Step 2: Selecting the animation speed by the speed progress bar which allows users to learn at their own preferred speed.

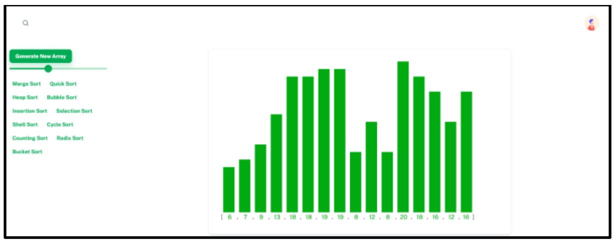

Fig. 9: Random Array bars generated after clicking the generate new array button.

Step 3: Once speed and random array is selected next step is select algorithm which user wants to visualize
Step 4: After clicking on the algorithm animation starts playing where the user can see how the selected algorithm is sorting the randomly generated array.

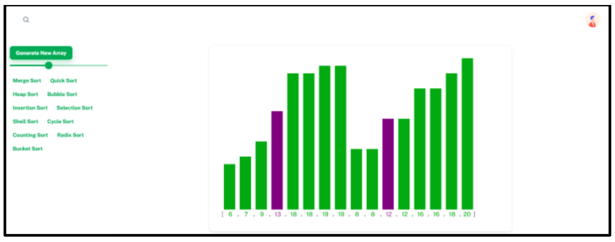

Fig. 10: Result taken during visualization of merge sort.

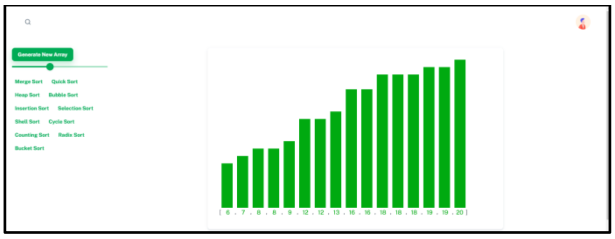

Fig. 11: Result taken after visualization of merge sort was completed

This is significant to see how the movement of data can change depending on what algorithms have already affected it. It can also lend a different perspective on how the algorithms perform on semi-sorted data compared to the given ordering options.

Table 1. Table for Time-Taken by the Tool to Complete Sorting Visualization

No. of elements in array	Algorithm visualization time (sec)					
	Bubble sort	Selection sort	Insertion sort	Quick sort	Merge sort	Heap sort
25	4.5	3.7	3	3.5	5	1.3
50	18.5	13.2	10	6.5	7.7	5.4
75	42.5	28.9	23.5	11	12.5	9.3
100	75	50	41	17	17.5	13.5

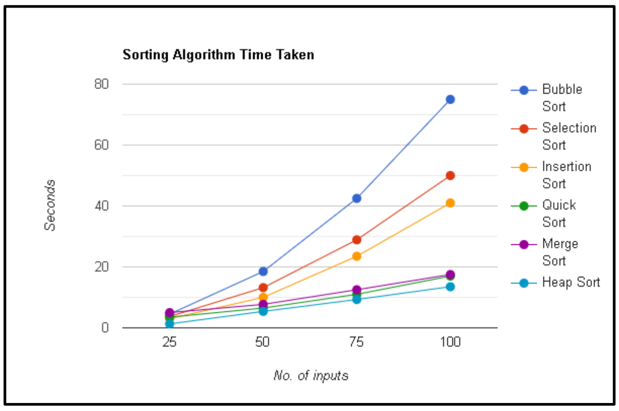

Fig. 12: Time-taken for visualizing sorting algorithms

Sorting Visualization tool can also be used to compare different algorithms and get useful insights as shown in Table 1.It shows time taken by each algorithm for visualization with respect to the number of array elements that were to be sorted. Theoretical time complexities can be harder to compare different algorithms. Thus, by comparing the visualization time taken by algorithms, users can get a better understanding of the performance of each algorithm. Further these data can visually be represented as shown in fig 12, Thus it is much easier to conclude from Fig 12 that heap sort was the best performer among other sorting algorithms considered during this experiment.

5.2 Interaction with Pathfinding visualization tool

The aim of the pathfinding visualization tool is to provide an interactive and highly user customizable playground to understand how different pathfinding algorithms work. Below are the steps how a user can use a pathfinding algorithm visualizer.

Step 1: Select the algorithm from the dropdown.
Step 2: Select the maze type. Once selected, users can see how the selected maze is being constructed on a 3D grid.
Step 3: There are additional configuration options like animation speed, resetting maze, setting up view which the user can do before visualization.
Step 4: Once all the configurations are done and maze-in constructed, the visualize button is enabled automatically by the system and the user can press it to start visualization. The dark blue colored node on the grid will represent the starting node and red node as the destination node. Once animation starts the user can view the grid in all three x, y and z planes. This gives an immersive 3D experience for the users. This animation makes sure to show how the algorithm is traversing different nodes in a maze and finding the shortest destination. In the figure 13 green colored nodes represent nodes visited during traversal for finding the shortest path.

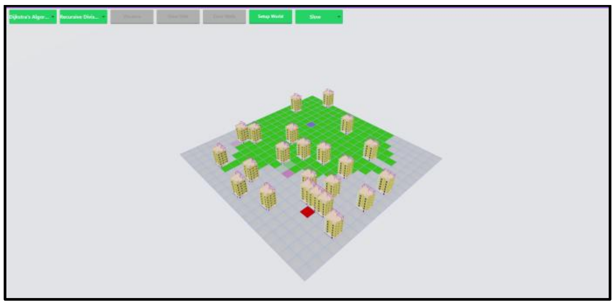

Fig. 13: Result taken during visualization of Dijkstra algorithm

Step 5: Once the traversing animation of nodes reaches the final node. Next phase of animation begins where backtracking is used to animate the path between start and final node. This final path will be colored differently as shown in fig 14.

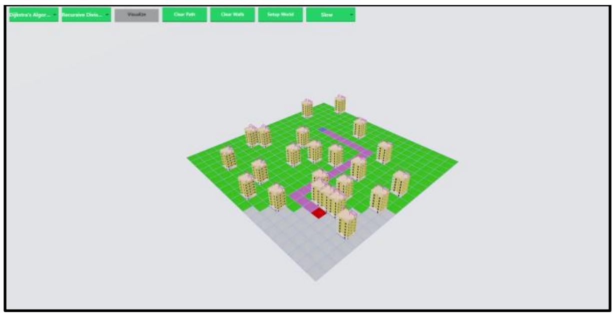

Fig. 14: Result taken after visualization of Dijkstra algorithm was completed

These animations and 3D grid can help users to play with different algorithms and compare each other in different constraints. 3D scenes and animation will help them relate these algorithms to real life applications.

Table 2. Table for time -taken by the tool to complete pathfinding visualization

Maze Type	Algorithm visualization time (sec)			
	Dijkstra	A-star	BFS	DFS
Random Maze	22.4	18.3	NA	NA
Recursive Maze	27	26	NA	NA
Maze with no weights	14.51	5.70	14.5	50.95
Accuracy of shortest path	99%	99%	99%	5%

Path finding algorithm visualization tool can also be used to analyze and compare performance of each algorithm as shown in table 2. For these analysis, Dijkstra and A-Star algorithm were visualized with slow animation speed in the same random and recursive maze on a grid of dimensions [20 , 20] where start node was at position [5 , 5] and destination node was at position [18 , 19] to calculate time taken to find and traverse to destination node from start position also at same time keeping track of accuracy of shortest path. From the results as shown in table 2 , it can be concluded that both algorithms find the shortest path with similar accuracy; however, A-star takes less time as compared to Dijkstra. Table 2 also shows performance of unweighted algorithms like Breadth first search (BFS) and Depth first search (DFS) algorithm with their time taken and accuracy of shortest path when there were no path blocks in the maze.

Below Table 3 shows performance analysis using different parameters of the 3D pathfinding visualization tool page in mobile and desktop by using Lighthouse. Lighthouse is an open source and automated tool for monitoring the quality of web pages. Following parameters were measured while calculating final performance score by lighthouse

- First Contentful Paint is the time taken for first image or text to paint
- Speed Index shows time taken to visibly populate content of the page
- Largest Contentful paint is time taken for largest image or text to paint
- Time to Interactive is time taken by page to become fully interactive
- Total Blocking Time sum of time period between first Contentful paint and time to interactive
- Cumulative Layout shift calculates movement of visible elements in viewport

Table 3. Table showing performance metrics given by lighthouse for the 3D Pathfinding Visualization tool web page

Performance parameter	Time		Weighting considered for calculating final performance score
	Desktop	Mobile	
First Contentful Paint	0.9s	3.5s	10%
Speed Index	2.4s	5.9s	10%
Largest Contentful Paint	1.4s	6.4s	25%
Time to interactive	1.3s	5.8s	10%
Total Blocking Time	60ms	330 ms	30%
Cumulative Layout Shift	0	0	15%

Based on results shown in Table 3 , final performance score given by lighthouse was 90 for the desktop and 55 for mobile out of 100 .According to lighthouse, Performance score in range 90-100 is considered best which is currently observed for 3D pathfinding visualization tool page when it loads on desktop devices. Since this tool uses WebGL libraries which use GPU power for executing graphical commands, a performance score of 55 for mobiles can still be considered good because of their lower end GPU.

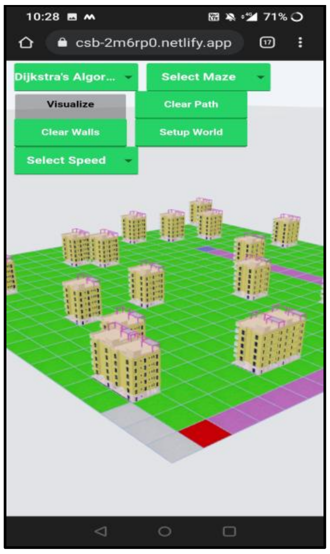

Fig. 15: Result taken through mobile for 3D pathfinding visualization tool

6. CONCLUSION

Sorting and path-finding visualization tools are successful in giving end users in depth knowledge and application of data structure and pathfinding visualization algorithms through 3D perspective and a platform where users can experiment and make analysis for different algorithm performance. Modern WebGL technology used for these tools like ThreeJS, React-three-Fiber and TweenJS helped efficiently to do 3D animations, render external 3D assets and good performance on lower end devices and mobiles. These tools can also be used to analyze and compare different algorithms. Based on the 3D visualization tool logs, it also showed in path finding algorithm, Dijkstra Algorithm, A-star, BFS, and DFS has 99%, 99%, 99%, and 05% respectively. Among these, A-star is most accurate and took less time to execute.Thus, being better than Dijkstra, depth first and breadth first search in most scenarios, In these way users can play in playground and do analysis by creating their own custom maze. Sorting algorithms are very useful for different scenarios. Imagine how much time it would take for a human to sort a list of thousands of names to make a phonebook, or a list of thousands of recipes, etc. Sorting visualization tools can help developers understand these algorithms more efficiently. Based on the sorting visualizer tool logs, it is found that heap sort is most efficient. Thus, this tool can be used to understand how algorithms work in different scenarios allowing them to learn and choose the right algorithm. Performance of the tool in mobile devices can be further improved like desktops by optimizing rendering of 3D assets. These tools can be further extended to animate more complex algorithms with unique animations, integrate VR for a more immersive maze experience which will help students and developers to analyze algorithms efficiently and relate them to real life use cases.

7. REFERENCES

[1] L. Li, X. Qiao, Q. Lu, P. Ren and R. Lin, "Rendering Optimization for Mobile Web 3D Based on Animation Data Separation and On-Demand Loading," in IEEE Access, vol. 8, pp. 88474-88486, 2020, doi: 10.1109/ACCESS.2020.2993613.

[2] G. Qiu and J. Chen, "Web-based 3D map visualization using WebGL," 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2018, pp. 759-763, doi: 10.1109/ICIEA.2018.8397815.

[3] D. B. Silva, R. d. L. Aguiar, D. S. Dvconlo and C. N. Silla, "Recent Studies About Teaching Algorithms (CS1) and Data Structures (CS2) for Computer Science Students," 2019 IEEE Frontiers in Education Conference (FIE), 2019, pp. 1-8, doi: 10.1109/FIE43999.2019.9028702.

[4] Tao Chen and T. Sobh, "A tool for data structure visualization and user-defined algorithm animation," 31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No.01CH37193), 2001, pp. TID-2, doi: 10.1109/FIE.2001.963845.

[5] C. Peng, "The research and design Of 3D Web guide system based On WebGL," The 26th Chinese Control and Decision Conference (2014 CCDC), 2014, pp. 4052-4054, doi: 10.1109/CCDC.2014.6852890.

[6] Z. He, M. Shi and C. Li, "Research and application of path-finding algorithm based on unity 3D," 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), 2016, pp. 1-4, doi: 10.1109/ICIS.2016.7550934.

[7] H. Kim, S. Nam, J. Park and D. Ko, "Direct canvas: Optimized WebGL rendering model," 2018 IEEE International Conference on Consumer Electronics (ICCE), 2018, pp. 1-3, doi:

10.1109/ICCE.2018.8326257.

[8] P. Li, X. Yu and J. Wang, "Progressive compression and transmission of 3D model with WebGL," 2016 International Conference on Audio, Language and Image Processing (ICALIP), 2016, pp. 170-173, doi: 10.1109/ICALIP.2016.7846665.

[9] J. Chen and W. Fang, "An Improved Spatial Data Structure for Web-based Large-scale 3D Scene," 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), 2019, pp. 681-684, doi: 10.1109/ICSESS47205.2019.9040768.

[10] H. Yutong, F. Wenlong and F. Yinshuang, "Research on model optimization based on 3DS max modeling," 2018 IEEE International Conference on Applied System Invention (ICASI), 2018, pp. 726-729, doi: 10.1109/ICASI.2018.8394362.

[11] Marek T., Krejcar O. (2015) Optimization of 3D Rendering in Mobile Devices. In: Younas M., Awan I., Mecella M. (eds) Mobile Web and Intelligent Information Systems. MobiWIS 2015. Lecture Notes in Computer Science, vol 9228. Springer, Cham. https://doi.org/10.1007/978-3-319-23144-0_4.