NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India
(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
☎: 08258 - 281039 – 281263, Fax: 08258 – 281265
Department of Computer Science and Engineering

Report on Mini Project

Simulation of Dijkstra's Algorithm

Course Code : 20CS607

Course Name: Computer Graphics

Semester: VI SEM Section: C

Submitted To:

Dr Sannidhan MS
Associate Professor
Department of Computer Science
and Engineering

Submitted By:

Prathiksha Kini – 4NM20CS139
Palguni Samaga – 4NM20CS127

Date of submission:

11-05-2023

Signature of Course Instructor

ABSTRACT

This report talks about our project titled “Dijkstra’s Algorithm” implemented using OpenGL. OpenGL provides a set of commands to render a three dimensional scene. That means you provide them in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL applications without licensing. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's.

TABLE OF CONTENTS

INTRODUCTION

This project is a simulation of Dijkstra's algorithm using the OpenGL graphics library. Dijkstra's algorithm is a well-known algorithm used for finding the shortest path between nodes in a graph with weighted edges. The algorithm works by starting at the source node and gradually expanding its search to other nodes in the graph. As the algorithm progresses, it maintains a distance table that keeps track of the shortest distance between the source node and each of the other nodes.

The purpose of this project is to provide a visual representation of how Dijkstra's algorithm works. Using OpenGL, the project displays a graphical representation of the input graph and animates the algorithm's progress as it searches for the shortest path. The input graph is represented by nodes and edges, and the user can interactively input the graph's details such as the number of vertices, shape of nodes, cost matrix, and source vertex. The output is displayed in a separate output area where the shortest path and the distance table are shown.

Overall, this project provides a fun and interactive way to understand Dijkstra's algorithm and its workings. By providing a visual representation of the algorithm's progress, users can better understand how the algorithm operates and the steps involved in finding the shortest path between nodes in a graph.

PROBLEM STATEMENT

The problem statement is to develop a simulation of Dijkstra's algorithm using OpenGL. Dijkstra's algorithm is a popular algorithm used to find the shortest path between nodes in a graph. The simulation will allow users to create a graph, specify start and end nodes, and then run the Dijkstra's algorithm to find the shortest path between the two nodes.

The simulation should be implemented using OpenGL, which is a powerful graphics library used to create interactive 3D applications. The simulation should allow users to interact with the graph by adding, removing, and modifying nodes and edges. The simulation should also display the progress of the algorithm, highlighting the nodes and edges that are currently being processed.

The simulation should provide visual feedback to the user, indicating the progress of the algorithm and the shortest path found. The shortest path should be highlighted in a distinctive color, making it easy for the user to identify.

Overall, the goal of the simulation is to provide a visual and interactive tool for users to learn and understand Dijkstra's algorithm and how it works to find the shortest path in a graph.

OBJECTIVES

The objectives of the simulation of Dijkstra's algorithm using OpenGL are:

1. To provide an interactive and visual tool for users to learn and understand Dijkstra's algorithm and its applications in finding the shortest path in a graph.

2. To allow users to create a graph and modify it by adding, removing, or modifying nodes and edges.

3. To simulate the Dijkstra's algorithm process and display the progress of the algorithm by highlighting the nodes and edges that are currently being processed.

4. To display the shortest path found by the algorithm and highlight it in a distinctive color, making it easy for the user to identify.

5. To provide a user-friendly interface that allows users to interact with the simulation easily.

6. To use OpenGL to create an attractive and immersive visualization of the graph and the algorithm process.

7. To enable users to experiment with different types of graphs, different start and end nodes, and different weights on the edges to see how they affect the shortest path found.

8. To provide an educational tool for students and researchers to learn and experiment with graph algorithms, specifically Dijkstra's algorithm.

HARDWARE / SOFTWARE Requirements

Hardware Requirements:

1. Processor-Intel or AMD (Advanced Micro Devices)
2. RAM-512MB (MINIMUM)
3. Hard Disk-1MB (MINIMUM)
4. Mouse
5. Keyboard
6. Monitor

Software Requirements:

1. OpenGL Libraries such has OpenGL Utility library, OpenGL Utility toolkit
2. C/C++ Language

METHODOLOGY

The Dijkstra algorithm is a popular algorithm used for finding the shortest path between nodes in a graph. To simulate this algorithm using OpenGL, the following methodology can be used:

1. Define a graph: The first step is to define the graph that needs to be traversed using the Dijkstra algorithm. This can be done by creating an adjacency matrix that represents the graph.

2. Create a visual representation of the graph: The next step is to create a visual representation of the graph. This can be done using OpenGL by creating nodes and edges using primitive shapes like circles and lines.

3. Initialize the algorithm: Initialize the algorithm by setting the source node and assigning a distance of 0 to it. All other nodes are assigned a distance of infinity.

4. Traverse the graph: The Dijkstra algorithm involves traversing the graph and updating the distances of adjacent nodes. This can be done using a loop that iterates through all the nodes in the graph.

5. Update distances: For each node, update the distance of adjacent nodes by checking if the current distance plus the weight of the edge is less than the current distance of the adjacent node.

6. Visualize the updates: As the algorithm progresses, the distances of nodes will be updated. This can be visualized using OpenGL by updating the colors or shapes of the nodes.

7. Highlight the shortest path: Once the algorithm is complete, highlight the shortest path between the source node and the destination node using OpenGL. This can be done by changing the color of the edges or nodes along the path.

Overall, the simulation of the Dijkstra algorithm using OpenGL involves creating a visual representation of the graph and updating the distances of nodes as the algorithm progresses. By highlighting the shortest path, the user can easily see the results of the algorithm.

IMPLEMENTATION

The project implements Dijkstra's algorithm simulation using OpenGL. The code begins by including the necessary header files, such as <stdio.h>, <iostream>, <GL/glut.h>, <math.h>, and <GL/gl.h>. The code also includes a few additional header files such as <windows.h> and <mmsystem.h>, which are used for playing sounds. The code defines various functions, including a raster function, a reverse function, a setFont function, a drawstring function, an initial function, and an itoa function. These functions are used for drawing text and lines on the OpenGL window. The code then defines the main function, which calls the title function and the read function. The title function is used for displaying the title page, and the read function is used for reading the input data from the user, such as the number of vertices, the cost matrix, and the source vertex. After the input is read, the code simulates Dijkstra's algorithm and displays the output on the OpenGL window. Finally, the code waits for the user to close the window before terminating.

1. **glRasterPos2f:** It is a function in OpenGL that sets the raster position for pixel operations in the current OpenGL context. It takes two arguments: the X and Y coordinates of the new raster position, specified as floating-point values.
2. **glutBitmapCharacter**: It is a function in the OpenGL Utility Toolkit (GLUT) library that is used to draw a single bitmap character in the current OpenGL context.
3. **setFont():** It is not a standard function in OpenGL, but it is commonly used in conjunction with other libraries such as GLUT or FreeGLUT to set the font for rendering text on the screen.
4. **itoa()** is a C++ function that converts an integer value to a null-terminated string using the specified base and stores the result in the array given by the second argument.
5. **glutInitDisplayMode :** Sets the initial display mode. The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.
6. **glutInitWindowPosition -** set the initial window position.
    - Declaration: void glutInitWindowPosition(int x, int y);

   x: Window X location in pixels. y: Window Y location in pixels.

7. **glutInitWindowSize –** Set the initial window size.
    • Declaration: void glutInitWindowSize(intwidth,int height);

   width: Width in pixels height: Height in pixels.
8. **glutCreateWindow** - Set the title to graphics window.
   • Declaration: glutCreateWindow(char *title);

   This function creates a window on the display. The string title can be used to label the window. The integer value returned can be used to set the current window when multiple windows are created.

.

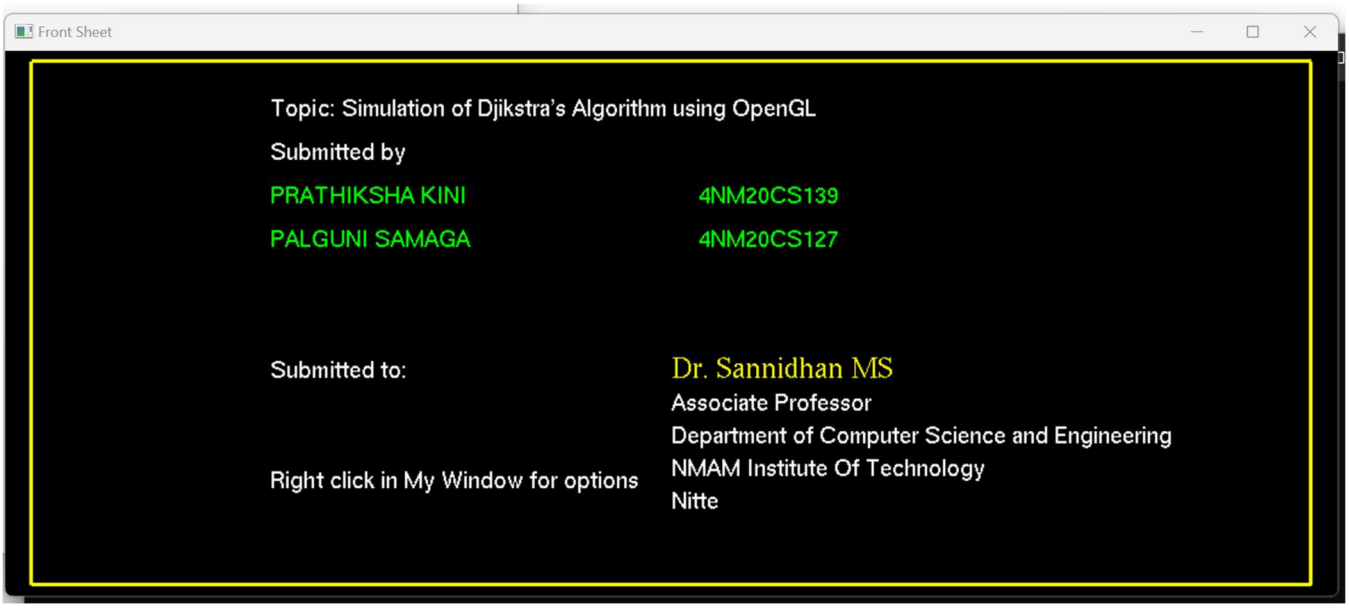
RESULTS AND DISCUSSIONS

The result of the program would be a simulation of Dijkstra's algorithm using OpenGL, which allows users to create a graph, specify start and end nodes, and run the algorithm to find the shortest path between the two nodes. The simulation would display the progress of the algorithm by highlighting the nodes and edges that are currently being processed, and display the shortest path found in a distinctive color.

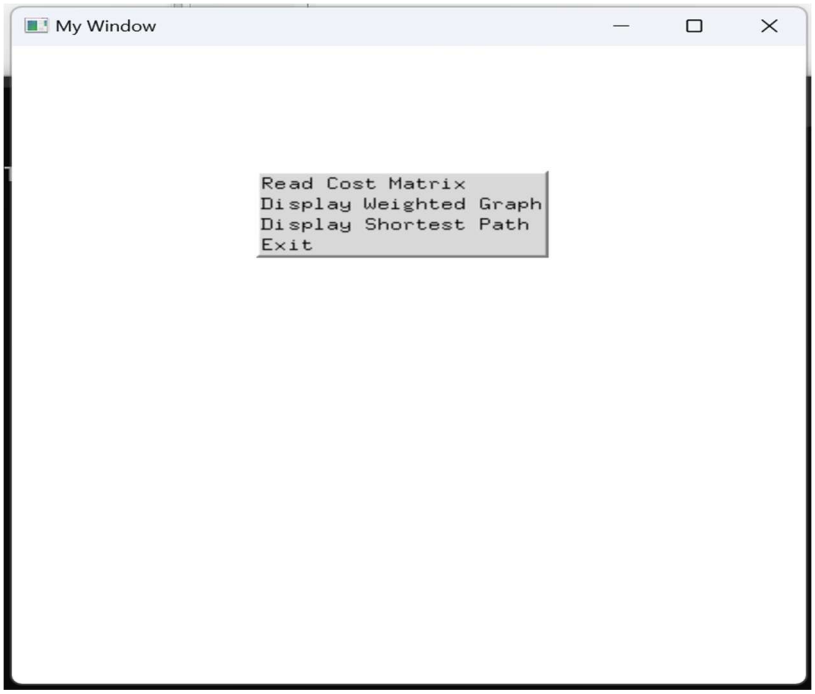
The program would provide a user-friendly interface that allows users to interact with the simulation easily. The user can modify the graph by adding, removing, or modifying nodes and edges. The user can experiment with different types of graphs, different start and end nodes, and different weights on the edges to see how they affect the shortest path found.

Overall, the program would provide an interactive and visual tool for users to learn and understand Dijkstra's algorithm and its applications in finding the shortest path in a graph. It would be an educational tool for students and researchers to learn and experiment with graph algorithms, specifically Dijkstra's algorithm.

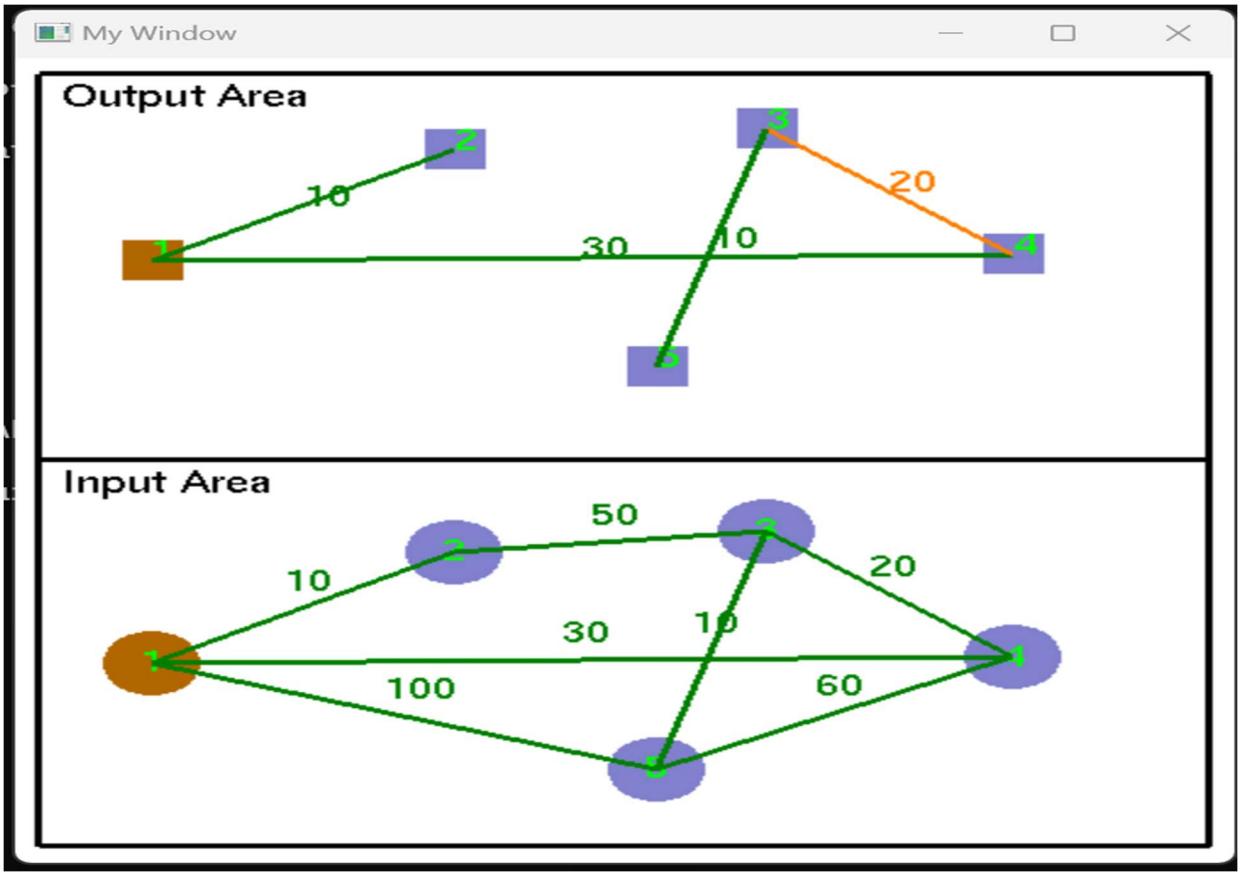
Screen Snapshots



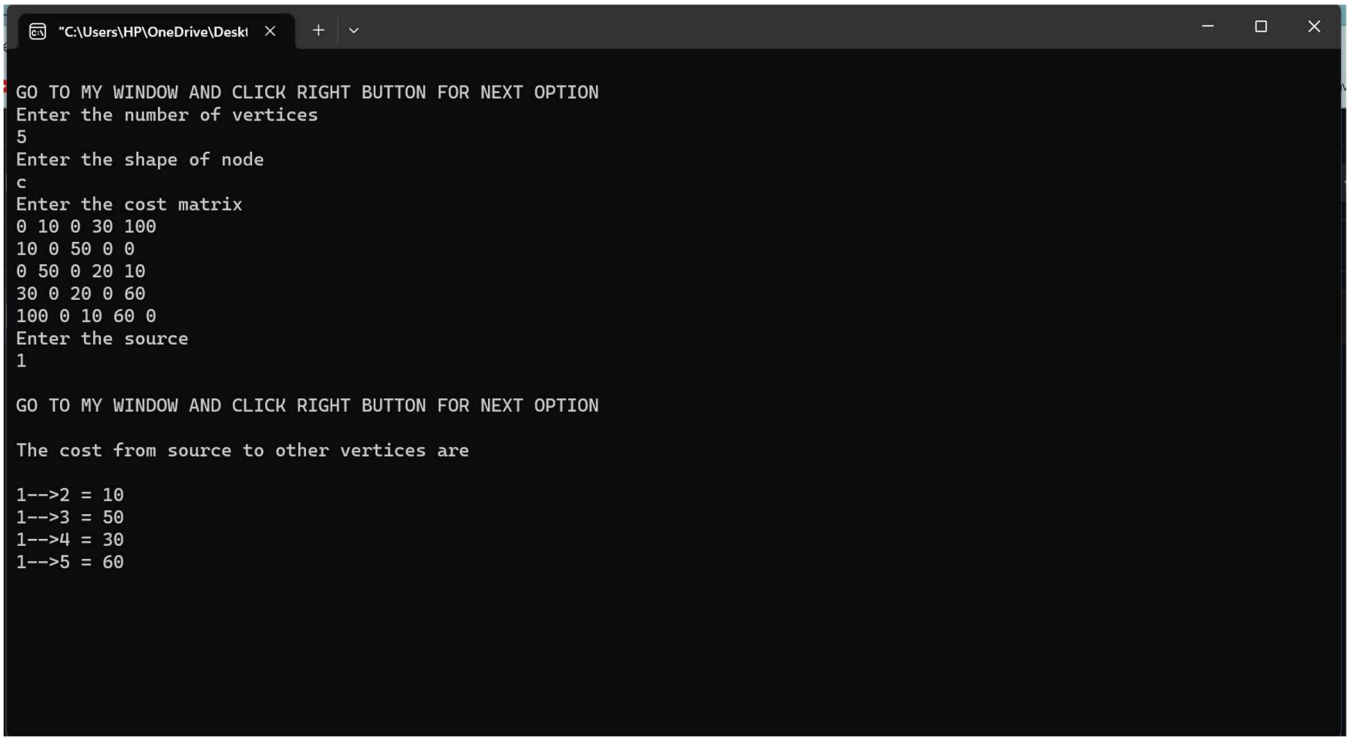
Front Page



Option Menu



Displaying Weighted graph and Shortest path



Output generated in the terminal

CONCLUSION AND FUTURE SCOPE

By implementing this project we got to know how to use some of the built in functions effectively and how to interact efficiently and easily. We got a good exposure of how algorithms and simulations are developed, by working on this project. The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C bindings for writing window system independent OpenGL programs. One of the major accomplishments in the specification of OpenGL was the isolation of window system dependencies from OpenGL's rendering model. The result is that OpenGL is window system independent. Window system operations such as the creation of a rendering window and the handling of window system events are left to the native window system to define. Necessary interactions between OpenGL and the window system such as creating and binding an OpenGL context to a window are described separately from the OpenGL specification in a window system dependent specification. The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT API (like the OpenGL API) is stateful. Most initial GLUT state is defined and the initial state is reasonable for simple programs. The GLUT routines also take relatively few parameters. No pointers are returned. The only pointers passed into GLUT are pointers to character strings (all strings passed to GLUT are copied, not referenced) and opaque font handles.

REFERENCES

1. “Computer Graphics with OpenGL” 4 th edition, Donald D. Hearn, M. Pauline Baker, Warren Carithers
2. http://www.opengl.org
3. http://www.openglprogramming.com
4. Q&A’s on Stack Overflow - https://stackoverflow.com/questions/tagged/opengl
5. OpenGL Installation Video CodeBlocks-20.03 Mingw || Installing CodeBlocks 20.30 || Installing FreeGlut || GLUT Tutorial - YouTube