

SIMULATION OF ARRAY OPERATIONS

Mini Project Report

Submitted to

Visvesvaraya Technological University

Belagavi-590018

By

MANISH J

USN:4SU18CS039

NITHIN THOMAS

USN:4SU18CS052

NIVIL SHIBU

USN:4SU18CS053

Under the guidance of

Mr. Pradeep G.S

Assistant Professor

In partial fulfilment of the requirement for the award of

VI Semester Computer Graphics Laboratory with Mini Project



Department of Computer Science and Engineering

SDM INSTITUTE OF TECHNOLOGY

UJIRE- 574240

2020-21

SDM INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi)

UJIRE – 574 240

Department of Computer Science and Engineering

CERTIFICATE

Certified that the project work titled ‘**Simulation of Array Operations**’ is carried out by **Mr. Manish J**, USN: **4SU18CS039**, **Mr. Nithin Thomas**, USN: **4SU18CS052**, **Mr. Nivil Shibu**, USN: **4SU18CS053** a bona-fide student of SDM Institute of Technology, Ujire, in partial fulfilment for the award of the Mini project on **Computer Graphics Laboratory with Mini Project** in Computer Science and Engineering of Visveswaraya Technological University, Belagavi during the year 2020-2021.

Mr. Pradeep G. S

Asst. Professor and Guide

Dr. Thyagaraju G. S

Professor and Head

External Viva

Name of the Examiners

Signature with Date

1.

2.

Acknowledgement

We express our deepest gratitude to our guide **Mr. Pradeep G S**, Asst. Professor, Department of Computer Science and Engineering, for his valuable guidance and encouragement while doing this project work. We are indebted to **Dr. Thyagaraju G S**, Head of the Department, and **Dr. Ashok Kumar T**, Principal, for their advice and suggestions at various stages of the work. We also extend our thanks to the management of SDM Institute of Technology, Ujire, for providing an excellent study environment, reference materials and laboratories facilities. We remain grateful to the co-operation and help rendered by the teaching and non-teaching staff of the department.

Manish J (4SU18CS039)

Nithin Thomas (4SU18CS052)

Nivil Shibu (4SU18CS053)

Abstract

Array Functions in C is a type of data structure that holds multiple elements of the same data type. The size of an array is fixed and the elements are collected in a sequential manner. There can be different dimensions of arrays and C programming does not limit the number of dimensions in an Array. Different operations of arrays: Insert, Delete, Search, Sort.

Table of contents

	Page No.
Acknowledgement	ii
Abstract	iii
Table of contents	iv
List of figures	v
Chapter 1 INTRODUCTION.....	1
Chapter 2 LITERATURE SURVEY.....	2
Chapter 3 ANALYSIS AND REQUIREMENT SPECIFICATIONS.....	7
3.1 Purpose.....	7
3.2 Scope.....	7
3.3 Functional Requirements.....	7
3.4 Non-Functional Requirements	8
Chapter 4 DESIGN.....	9
Chapter 5 IMPLEMENTATION.....	12
Chapter 6 SNAPSHOTS.....	27
Chapter 7 CONCLUSION.....	31
Chapter 8 FUTURE ENHANCEMENT.....	32
REFERENECES.....	33
PERSONAL PROFILE.....	34

List of Figures

	Page No.
Figure 4.1 Flow Chart.....	11
Figure 6.1 Title Page.....	27
Figure 6.2 Main Menu.....	27
Figure 6.3 Help page.....	28
Figure 6.4 Array insertion and Sub menu	28
Figure 6.5 Array Search.....	29
Figure 6.6 Array Deletion.....	29
Figure 6.7 Array Sort.....	30
Figure 6.8 Array updation.....	30

INTRODUCTION

1. Searching

The search operation is used to find a particular data item or element in an Array. We can perform searching in an unsorted array with the help of traversal of the Array. The linear traversal from the first element to the last element can be used to search if a given number is present in an Array and can also be used to find its position if present.

2. Insertion

Insertion operation is used to add a new element in the Array. When we specify the particular element and position where it is to be added in the Array, we perform insertion operation. However, the size of the Array is not disturbed while performing this operation. An element will be inserted in an array only if it has sufficient space to add it. If the size of an array is full already, a new element cannot be added. An example to show insert operation in an unsorted Array in C.

3. Deletion

In delete operation, the element which already exists in the Array is searched (using linear search) and deleted, followed by the shifting of elements. The user enters the position of the element which is to be deleted from the array. Deletion operation, just like the insertion operation, does not affect the size of array. Also, the position of the element to be deleted should be within the size of array, since the deletion of an element beyond the size of Array is not possible. C program to show delete operation in an unsorted array.

4. Sorting

This operation is performed to sort an Array into a fixed order, i.e., either ascending or descending. Here is an example of sort operation on an Array in C.

LITERATURE SURVEY

2.1 Overview Computer Graphics

Computer graphics is a sub-field of Computer Science which studies methods for digitally synthesizing and manipulating visual content. Although the term often refers to the study of three-dimensional computer graphics, it also encompasses two-dimensional graphics and image processing. Computer graphics is concerned with all aspects of producing pictures or images using a computer. The field began humbly almost 50 years ago, with the display of few lines on Cathode Ray Tube (CRT). We can create images by computer that are indistinguishable from photographs of real objects. We routinely train pilots with simulated airplanes, generating graphical displays of a virtual environment in real time. Feature-length movies made entirely by computer have been successful, both critically and financially. Massive multiplayer games can involve tens of thousands of concurrent participants. OpenGL, a graphics software system has become a widely accepted standard for developing graphics applications. Fortunately, OpenGL is easy to learn and it possesses most of the characteristics of other popular graphics system. Computer graphics studies the manipulation of visual and geometric information using computational techniques.

2.1.1 Applications of Computer Graphics

The development of computer graphics has been driven both by the needs of the user community and by the advances in hardware and software. The applications of computer graphics are many and varied; it can be divided into four major areas:

- Display of information
- Design
- Simulation and animation
- User interfaces

Display of Information

It is human's ability to recognize visual patterns that ultimately allows us to interpret the information contained in the data. The field of information visualization is becoming

SIMULATION OF ARRAY OPERATIONS

increasingly more important. Modern imaging technologies –such as computed tomography (CT), magnetic resonance imaging (MRI), ultrasound, and positron –emission tomography (PET)-generate three-dimensional data that must be subjected to algorithmic manipulation to provide useful information.

Design

Professions such as engineering and architecture are concerned with design. Starting with a set of specification engineers and architects seek a cost effective and aesthetic solutions that satisfies the specifications. Design is an iterative process rarely in the real world is a problem specified such that there is a unique optimal solution. Design problems are either over determined such that they possess no solution that satisfies all the criteria, much less than optimal solution or underdetermined such that they have multiple solution that satisfy the design criteria. Thus, the designer works iteratively.

Simulation and Animation

Once graphic systems evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. One of the most important users has been in the training of pilots. Graphical flite simulators have proved to increase safety and to reduce training expenses. The use of special VLSI chips has led to a generation of arcade games as sophisticated as the flight.

User Interfaces

Our interactions with computers have become dominated by a visual paradigm that includes windows, icons, menus, pointing device, such as a mouse. Although we are familiar with the style of graphical user interface used on mouse work stations, advances in computer graphics have made possible other forms of interfaces.

2.2 Overview of OpenGL

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed to work efficiently even if the computer that displays the graphics you create isn't the computer that runs your graphics program. This might be the case if you work in a networked computer environment where many computers are connected to one another by wires capable of

SIMULATION OF ARRAY OPERATIONS

carrying digital data. In this situation, the computer on programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running which your program runs and issues OpenGL drawing commands is called the client, and the computer that receives those commands and performs the drawing is called the server. The format for transmitting OpenGL commands (called the protocol) from the client to the server is always the same, so OpenGL across a network, then there's only one computer, and it is both the client and the server. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitive-points, lines, and polygons.

2.2.1 OpenGL Command Syntax

As you might have observed from the simple program in the previous section, OpenGL commands use the prefix `gl` and initial capital letters for each word making up the command name (recall `glClearColor()`, for example). Similarly, OpenGL defined constants begin with `GL_` use all capital letters, and use underscores to separate words (like `GL_COLOR_BUFFER_BIT`). You might also have noticed some seemingly extraneous letters appended to some command names (the `3f` in `glColor3f()`, for example). It's true that the `Color` part of the command name is enough to define the command as one that sets the current color.

2.2.2 OpenGL related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Therefore, you might want to write your own library on top of OpenGL to simplify your programming tasks. Also, you might want to write some routines that allow an OpenGL program to work easily with your windowing system. In fact, several such libraries and routines have already been written to provide specialized features, as follows. Note that the first two libraries are provided with every OpenGL implementation, the third was written for this book and is available using ftp, and the fourth is a separate product that's based on OpenGL. The OpenGL Utility

SIMULATION OF ARRAY OPERATIONS

Library (GLU) contain several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of your OpenGL implementation. It's described in more detail in Appendix C and in the OpenGL Reference Manual. The more useful GLU routines are described in the chapters in this guide, where they're relevant to the topic being discussed. GLU routines use the prefix `glu`. The OpenGL Extension to the X window system (GLX) provides a means of creating an OpenGL context and associating it with a drawable window on a machine that uses the X Window System. GLX is provided as an adjunct to OpenGL. The OpenGL Programming Guide Auxiliary library was written specifically for this book to make programming examples simpler and yet more complete. It's the subject of the next section, and it's described in more detail in Appendix E. Auxiliary library routines use the prefix `aux`. "How to obtain the sample code" describes how to obtain the source code for the auxiliary library.

2.2.3 OpenGL Primitives

A primitive is an interpretation of a list of vertices into a certain shape. In OpenGL, we specify the list of vertices for a certain primitive by placing them in a `glBegin(); glEnd();` block as follows:

- `glBegin(type);`
- `glVertex(..);` and `glEnd();`

The type can be one of ten symbolic constants:

- `GL_POINTS`
- `GL_LINES`
- `GL_LINE_STRIP`
- `GL_LINE_LOOP`
- `GL_TRIANGLES`
- `GL_TRIANGLE_STRIP`
- `GL_TRIANGLE_FAN`
- `GL_QUADS`
- `GL_QUAD_STRIP` and `GL_POLYGON`

2.3 Overview of Arrays

An array is a fixed-length data structure that can contain multiple objects of the same type. An array can contain any type of object, including arrays. To declare an array, you use the type of object that the array can contain and brackets. The length of the array must be specified when it is created. You can use the new operator to create an array, or you can use an array initializer. Once created, the size of the array cannot change. To get the length of the array, you use the length attribute. An element within an array can be accessed by its index. Indices begin at 0 and end at the length of the array minus 1.

ANALYSIS AND REQUIREMENT SPECIFICATIONS

3.1 Purpose

In the project “Simulation of Array Operations”, the computer graphics concepts along with the OpenGL functions are used. Here in this mini-project we demonstrate the simulation of array operations puzzle which helps the user to understand how the operations happen behind the scenes. The purpose of this document is to explain the functionality of the animated application. The study of requirement specification is focused specifically on the functioning of the system. It allows the developer/analyst to understand the system, function to be carried out, performance levels to be obtained and corresponding interfaces to be established.

3.2 Scope

The scope of the project is to portray a 3D environment that provides the user with the example of various basic transformations and shading techniques available in OpenGL. It provides most of the features that a 3-D graphics editor should have. It is developed in C++. It has been implemented in LINUX platform.

3.3 Functional Requirements

Some of the function that are used in this project are:

- `glClear();`
- `glClearColor();`
- `glColor();`
- `glutInit();`
- `glutInitDisplayMode();`
- `glutDisplayFunc();`
- `glutWindowPosition();`
- `glutMouseFunc();`
- `glutKeyboardFunc();`
- `tower();`

SIMULATION OF ARRAY OPERATIONS

- `glTranslatef();`
- `glRotatef();`
- `glutMainLoop();`

3.4 Non-Functional Requirements

Hardware Specification

Processor	: i3 Core Processor.
Clock Speed	: 2.20GHz.
Monitor	: 1024*768 Resolution Colour.
RAM	: 1GB
Hard Disk	: 40GB
Keyboard	: QWERTY

Input output console for interaction.

Software Specification

Operating System	: Ubuntu16.04 & Higher
OpenGL Libraries	
C/C++ platforms	

DESIGN

Requirement analysis is an important part of the design process. Here we identify the needs or requirements. Once the requirements have been identified the solution for the requirement can be designed. This project has been developed using OpenGL function which is function oriented. Changes at the function and the way in which it uses a system state may cause anticipated changes in the behaviour of other functions. A functional approach to design is therefore most likely to be successful when the amount of system stated information is minimized and information sharing is explicit. so the implementation technique used is more efficient and compact.

4.1 Module Used

1) **int main()**

Execution of program starts here. All the call back functions are called here.

2) **frontscreen()**

This function is used to display the title page.

3) **display()**

This function is used to display the page of TBT implementation.

4) **display2()**

This function is used to display the menu page.

5) **drawstring()**

This function is used to draw the string.

6) **void display()**

The function basically displays all the above-described functions on the screen as we flush the output onto the screen from frame buffer.

4.2 Functions

4.2.1 glBegin, glEnd Function

The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives. These primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. The following are accepted symbolic constants

SIMULATION OF ARRAY OPERATIONS

and their meanings.

4.2.2 Transformations

Scaling Function

Scaling is an affine non-rigid body transformation body by which you can make an object bigger or smaller.

```
glScalef(x value, y value ,z value);
```

Translate Function

Is an operation that displaces points by a fixed distances in a given direction.

```
glTranslatef(x value, y value, z value);
```

4.2.3 Colouring

The objects are given different colours to improve the looks of the project. For example,

```
glColor3f(RGB value);
```

It is possible to apply background colouring as shown below.

```
glClearColor(1.0, 1.0, 1.0, 1.0); // for white background
```

```
glClearColor(0.0, 0.0, 0.0, 0.0); // for black background
```

4.2.4 Callback Function

- **glutDisplayFunc():**

glutDisplayFunc sets the display call-back for the current window.

- **glutKeyboardFunc():**

Register the keyboard call-back function. The call-back function returns the ASCII code of the key pressed and the position of the mouse.

```
void keys(unsigned char key, int x, int y) ;
```

- **glutMouseFunc():**

Register the mouse call-back function. The call-back function returns the button (GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON). The state of the button after event (GLUT_UP, GLUT_DOWN) and the position of the mouse with respect to top left of the window.

```
Void mouse(int btn, int state, int x, int y);
```

SIMULATION OF ARRAY OPERATIONS

4.2.5 glutInit Function

glutInit is used to initialize the GLUT library.

4.2.6 glutInitDisplayMode Function

glutInitDisplayMode sets the initial display mode. Display mode, normally the bitwise OR-ing of GLUT display mode bit masks.

- GLUT_RGB is alias for GLUT_RGB.
- GLUT_DOUBLE is bit mask to select double buffered window.
- GLUT_DEPTH is bit mask to select a window with depth buffer.

4.2.7 glutMainloop Function

glutMainloop enters the GLUT event processing loop.

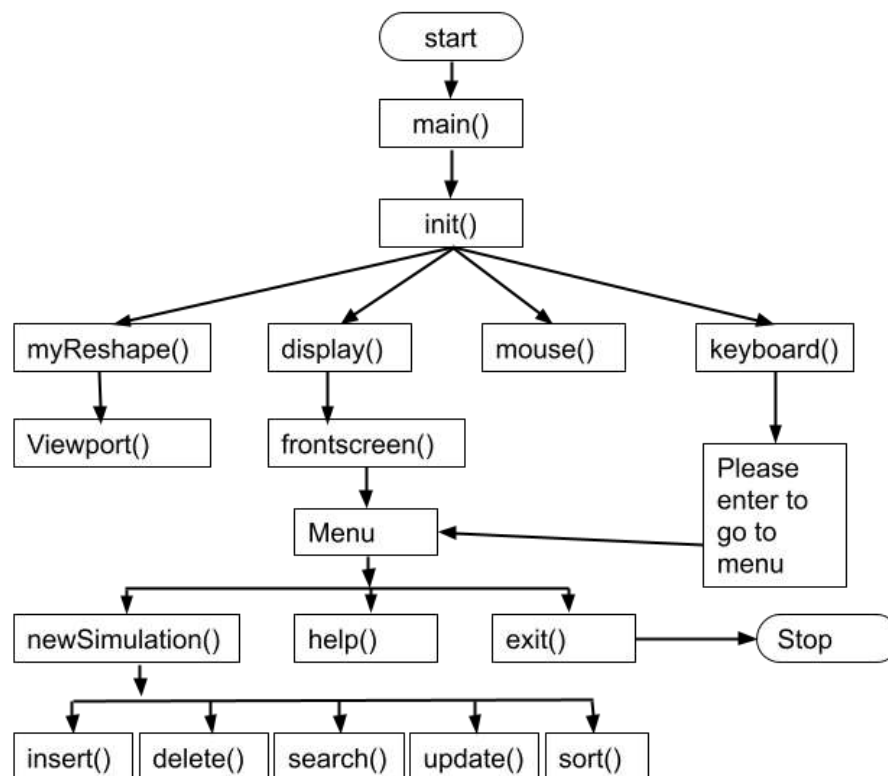


Figure 4.1: Flow Chart

IMPLEMENTATION

5.1 Implementation of Main window

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

    glutInitWindowSize(1500, 500);

    glutInitWindowPosition(0, 0);

    glutCreateWindow("Array");

    glutCreateMenu(array_menu);

    glutAddMenuEntry("Insert", 1);

    glutAddMenuEntry("Delete", 2);

    glutAddMenuEntry("search ", 3);

    glutAddMenuEntry("Update", 4);

    glutAddMenuEntry("Sort", 5);

    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glutReshapeFunc(myReshape);

    glutKeyboardFunc(keys);

    glutMouseFunc(mouse);

    glutDisplayFunc(display);

    glEnable(GL_DEPTH_TEST);

    glClearColor(1.0, 1.0, 1.0, 1.0);
```

SIMULATION OF ARRAY OPERATIONS

```
glutMainLoop();
```

```
return 0; }
```

5.2 Implementation of Reshape function

```
void myReshape(int w, int h)
```

```
{
```

```
glViewport(0, 0, w, h);
```

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

```
if (w <= h)
```

```
glOrtho(-4.0, 4.0, -4.0*(GLfloat)h/(GLfloat)w, 4.0*(GLfloat)h/(GLfloat)w, -4.0, 4.0);
```

```
else
```

```
glOrtho(-4.0*(GLfloat)w/(GLfloat)h, 4.0*(GLfloat)w/(GLfloat)h, -4.0, 4.0, -4.0, 4.0);
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glutPostRedisplay();
```

```
}
```

5.3 Implementation of Title page

```
void displayText(float x, float y, float R1, float G1, float B1, const char* string) {
```

```
int j = strlen(string);
```

```
glColor3f(R1, G1, B1);
```

```
glRasterPos2f(x, y);
```

```
for (int i = 0; i < j; i++)
```

```
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
```

```
glFlush();
```

```
}
```

SIMULATION OF ARRAY OPERATIONS

```
void frontend()
{
    displayText(-2.3, 3.5, 0, 0, 1, "SDM INSTITUTE OF TECHNOLOGY");
    displayText(-3.3, 3, 0, 1, 0.1, "DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING");
    displayText(-1.8, 2.5, 1, 0.5, 1.0, "MINI PROJECT ON");
    displayText(-2.8, 1.7, 0, 1, 1, "SIMULATION OF ARRAY OPERATIONS");
    displayText(-6, 00, 0.6, 0.3, 0.9, "SUBMITTED BY");
    displayText(-6, -0.5, 0.6, 0.3, 0.9, "NIVIL SHIBU");
    displayText(-6, -0.9, 0.6, 0.3, 0.9, "(4SU18CS053)");
    displayText(-6, -1.6, 0.6, 0.3, 0.9, "NITHIN THOMAS");
    displayText(-6, -2.1, 0.6, 0.3, 0.9, "(4SU18CS052)");
    displayText(-6, -2.5, 0.6, 0.3, 0.9, "MANISH J");
    displayText(-6, -2.8, 0.6, 0.3, 0.9, "(4SU18CS039)");
    displayText(3.5, 00, 0.6, 0.3, 0.9, "GUIDED BY");
    displayText(3.5, -0.5, 0.6, 0.3, 0.9, "PRADEEP G S");
    displayText(3.5, -0.9, 0.6, 0.3, 0.9, "Asst. Prof Dept of CSE");
    displayText(-1, -3.1, 0.6, 0.3, 0.9, "[Press Enter to Start]");
    glFlush();
}
```

5.4 Implementation of Menu page

```
void buttonm() {
    displayText(-1.2, 2.2, 1.0, 1.0, 1.0, "New Simulation");
    glColor3f(1.0, 0.5, 0.5);
```

SIMULATION OF ARRAY OPERATIONS

```
glBegin(GL_POLYGON);

glVertex2f(-1.5, 2);

glVertex2f(-1.5, 2.5);

glVertex2f(0.5, 2.5);

glVertex2f(0.5, 2);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glLineWidth(6.5f);

glBegin(GL_LINES);

glVertex2f(0.5, 2);

glVertex2f(-1.53, 2);

glVertex2f(-1.5, 2);

glVertex2f(-1.5, 2.5);

glEnd();

displayText(-0.8, 0.7, 1.0, 1.0, 1.0, "Help");

glColor3f(1.0, 0.5, 0.5);

glBegin(GL_POLYGON);

glVertex2f(-1.5, 0.5);

glVertex2f(-1.5, 1);

glVertex2f(0.5, 1);

glVertex2f(0.5, 0.5);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glLineWidth(6.5f);
```

SIMULATION OF ARRAY OPERATIONS

```
glBegin(GL_LINES);

glVertex2f(0.5, 0.5);

glVertex2f(-1.53, 0.5);

glVertex2f(-1.5, 0.5);

glVertex2f(-1.5, 1);

glEnd();

displayText(-0.8, -0.8, 1.0, 1.0, 1.0, "Exit");

glColor3f(1.0, 0.5, 0.5);

glBegin(GL_POLYGON);

glVertex2f(-1.5, -0.5);

glVertex2f(-1.5, -1);

glVertex2f(0.5, -1);

glVertex2f(0.5, -0.5);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glLineWidth(6.5f);

glBegin(GL_LINES);

glVertex2f(-1.5, -0.5);

glVertex2f(-1.5, -1);

glVertex2f(-1.53, -1);

glVertex2f(0.5, -1);

glEnd();

}

void display()
```

SIMULATION OF ARRAY OPERATIONS

```
{  
  
glClearColor(R, G, B, 0.0);  
  
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
  
switch (l)  
{  
  
case 0:  
  
    fullscreen();  
  
    break;  
  
case 1:  
  
    buttonm();  
  
    break;  
  
case 2:  
  
    node_vertex();  
  
    break;  
  
case 3:  
  
    help();  
  
    break;  
  
case 4:  
  
    node_vertex();  
  
    break;  
  
}  
  
glFlush();  
  
glFlush(); }
```

5.4 Implementation of Array operations

SIMULATION OF ARRAY OPERATIONS

```
void keys(unsigned char key, int x, int y) {  
  
    if (key == 'c' || key == 'C')  
  
    {  
  
        if (f1 == 1) {  
  
            R = 0.0666; G = 0.0588; B = 0.4313;  
  
            f1 = 2;  
  
        }  
  
        else if (f1 == 2) {  
  
            R = 0.3686; G = 0.0; B = 0.5803;  
  
            f1 = 3;  
  
        }  
  
        else if (f1 == 3) {  
  
            R = 0.0078; G = 0.4313; B = 0.5607;  
  
            f1 = 1;  
  
        }  
  
    }  
  
    if (key == 13) {  
  
        l = 1;  
  
    }  
  
    if (key == 27)  
  
    {  
  
        if (l == 3)  
  
        {  
  
            l--;  
  
        }  
  
    }  
}
```


SIMULATION OF ARRAY OPERATIONS

```
}

if (l == 4)

{

l = l - 2;

}

l--;

}

display();

}

void draw_node(float i, float j, int ch1, int mv)

{

if (l > 0)

{

if (ch1 == 0)

{

char text[20];

int vv = 0;

sprintf_s(text, "%d", mv);

glColor3f(1.0, 0.0, 0.0);

glRasterPos2f(i + 0.1, j - 0.28);

while (text[vv] != '\0')

{

glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[vv]);

vv++;

}
```

SIMULATION OF ARRAY OPERATIONS

```
}

glColor3f(1.0, 0.60, 0.0);

glBegin(GL_LINES);

glVertex2f(i, 0);

glVertex2f(i, j);

glVertex2f(i, j);

glVertex2f(i + 0.5, j);

glVertex2f(i + 0.5, j);

glVertex2f(i + 0.5, 0);

glVertex2f(i + 0.5, 0);

glVertex2f(i, 0);

glEnd();

}

else

{

char text[20];

int vv = 0;

sprintf_s(text, "%d", mv);

glColor3f(0.0, 1.0, 0.0);

glRasterPos2f(i + 0.1, j - 0.28);

while (text[vv] != '\0')

{

glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[vv]);

vv++;
```

SIMULATION OF ARRAY OPERATIONS

```
}

glColor3f(0.0, 1.0, 0.0);

glBegin(GL_LINES);

glVertex2f(i, 0);

glVertex2f(i, j);

glVertex2f(i, j);

glVertex2f(i + 0.5, j);

glVertex2f(i + 0.5, j);

glVertex2f(i + 0.5, 0);

glVertex2f(i + 0.5, 0);

glVertex2f(i, 0);

glEnd();

}

}

else

{

glColor3f(1.0, 1.0, 1.0);

glBegin(GL_POLYGON);

glVertex2f(i, 0);

glVertex2f(i, j);

glVertex2f(i + 0.5, j);

glVertex2f(i + 0.5, 0);

glEnd();

}
```

SIMULATION OF ARRAY OPERATIONS

```
glFlush();

}

void node_vertex(void)

{

int c = -1;

li = -6.5;

if (n == -1) {

l = 0;

draw_node(1, 1, 0, 1);

}

else {

printf("The Data present in Linked List are: ");

for (int r = 0; r <= n; r++)

{

li = li + 0.5;

}

for (float i = -6; i <= li; i += 0.5)

{

ch1 = 0;

c++;

if (ar[c] == se)

{

ch1 = 1;

}

}
```

SIMULATION OF ARRAY OPERATIONS

```
draw_node(i, j, ch1, ar[c]);

}

}

}

void array_menu(int choice)

{

int va, va2;

switch (choice)

{

case 1:

n++;

printf("Enter the data: ");

scanf_s("%d", &ar[n]);

l = 2;

display();

break;

case 2:

ch = 0;

printf("Enter the data: ");

scanf_s("%d", &de);

for (int r = 0; r <= n; r++)

{

if (de == ar[r] || ch == 1)

{
```

SIMULATION OF ARRAY OPERATIONS

```
ar[r] = ar[r + 1];

ch = 1;

}

}

if (ch == 1)

{

n--;

}

if (n == -1)

{

l = 0;

}

display();

break;

case 3:

se = 0;

printf("Enter the data: ");

scanf_s("%d", &de);

for (int r = 0; r <= n; r++)

{

if (ar[r] == de)

{

se = de;

}

}
```

SIMULATION OF ARRAY OPERATIONS

```
}

display();

break;

case 4:

printf("Enter the array index: ");

scanf_s("%d", &de);

for (int r = 0; r <= n; r++)

{

if (r == de - 1)

{

printf("Enter the value: ");

scanf_s("%d", &ar[r]);

}

}

display();

break;

case 5:

for (f1 = 0; f1 < n; f1++) {

f3 = f1;

for (f2 = f1 + 1; f2 <= n; f2++)

{

if (ar[f2] < ar[f3])

{

f3 = f2;
```

SIMULATION OF ARRAY OPERATIONS

```
}  
  
}  
  
temp = ar[f1];  
ar[f1] = ar[f3];  
ar[f3] = temp;  
  
}  
  
for (int r = 0; r <= n; r++)  
{  
    printf(" %d", ar[r]); }  
  
display();  
  
break;  
  
case 6:  
    exit(0); } }
```


SNAPSHOTS



Figure 6.1: Title Page

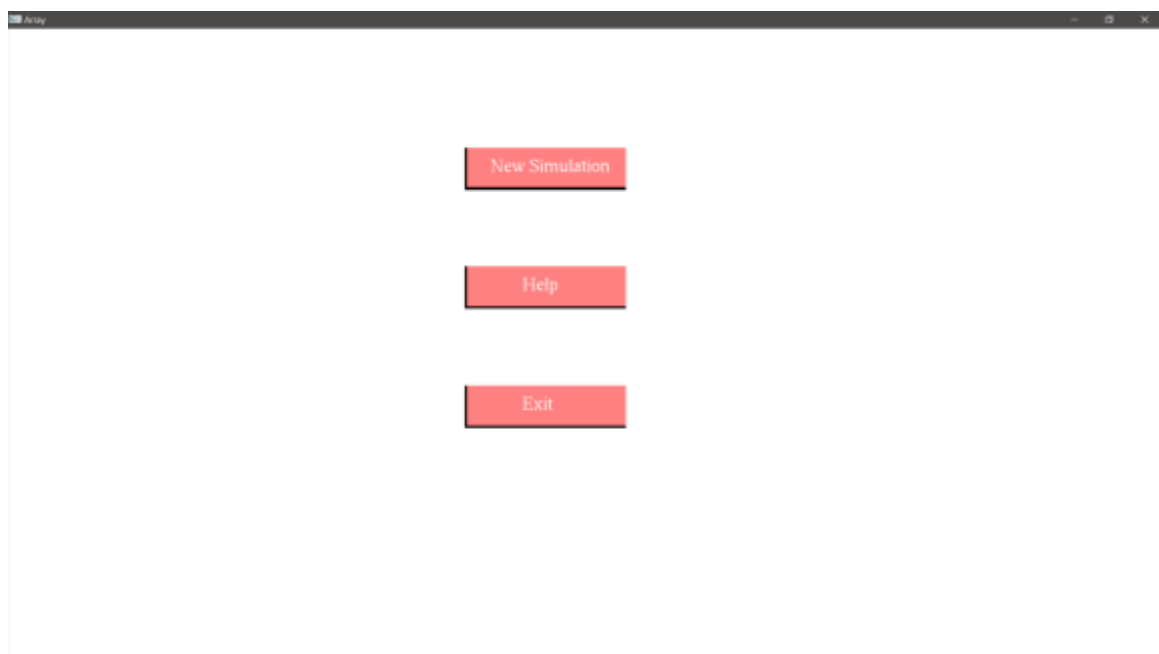


Figure 6.2: Main Menu

SIMULATION OF ARRAY OPERATIONS

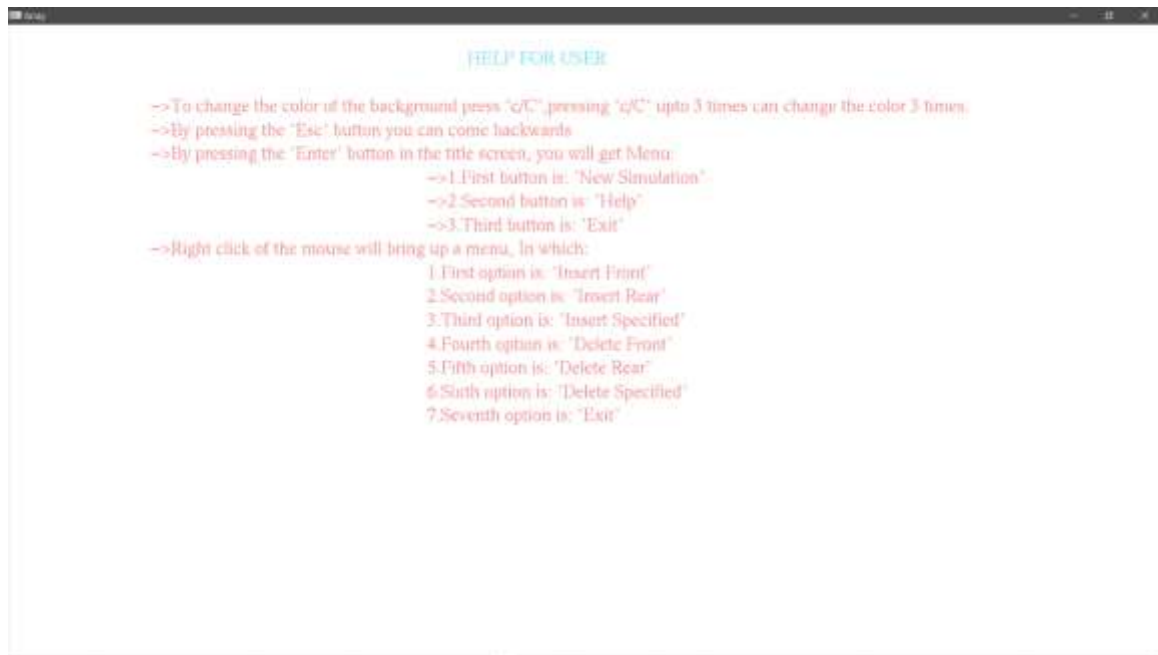


Figure 6.3: Help page



Figure 6.4: Array insertion and Sub menu

SIMULATION OF ARRAY OPERATIONS



Figure 6.5: Array Search

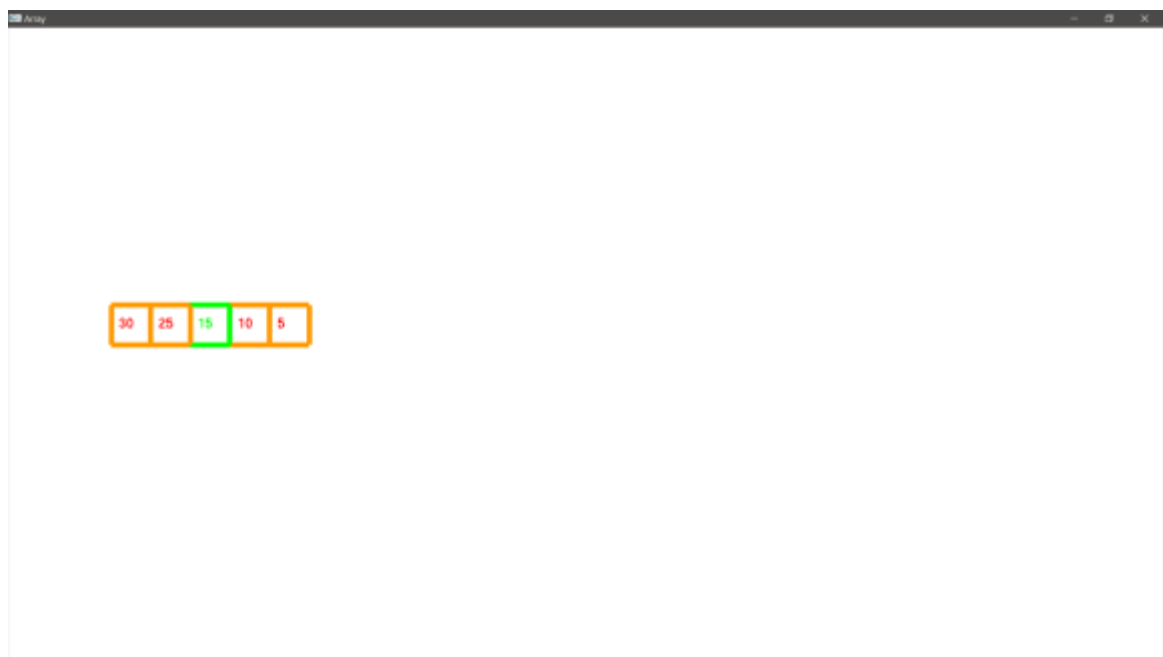


Figure 6.6: Array Deletion

SIMULATION OF ARRAY OPERATIONS

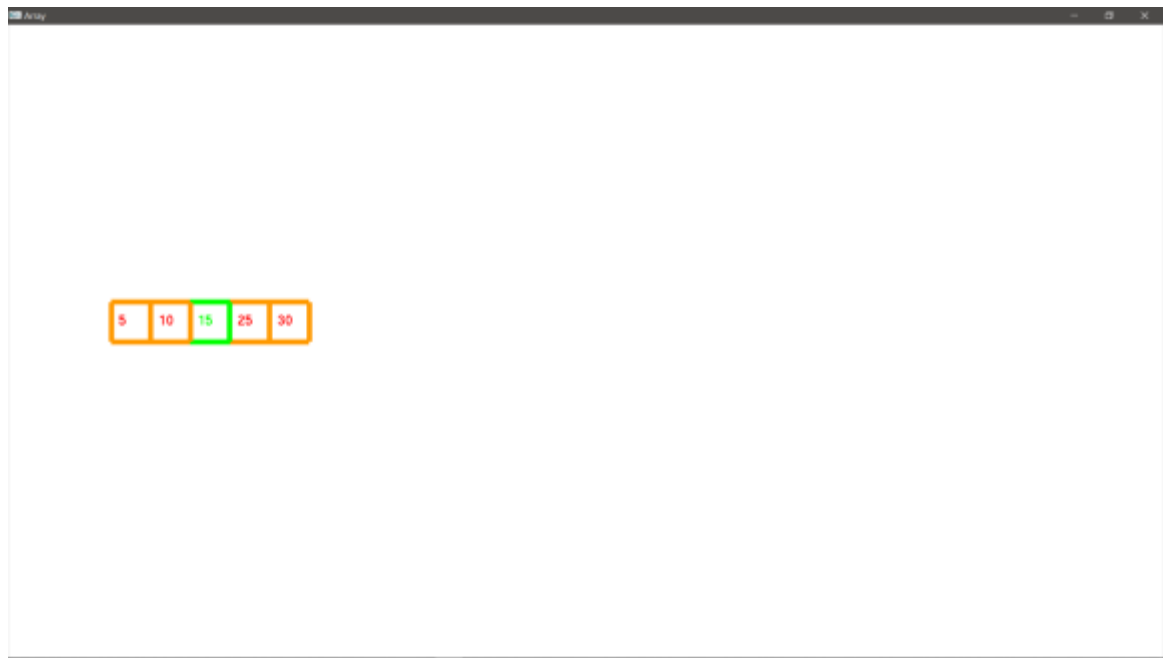


Figure 6.7: Array Sort

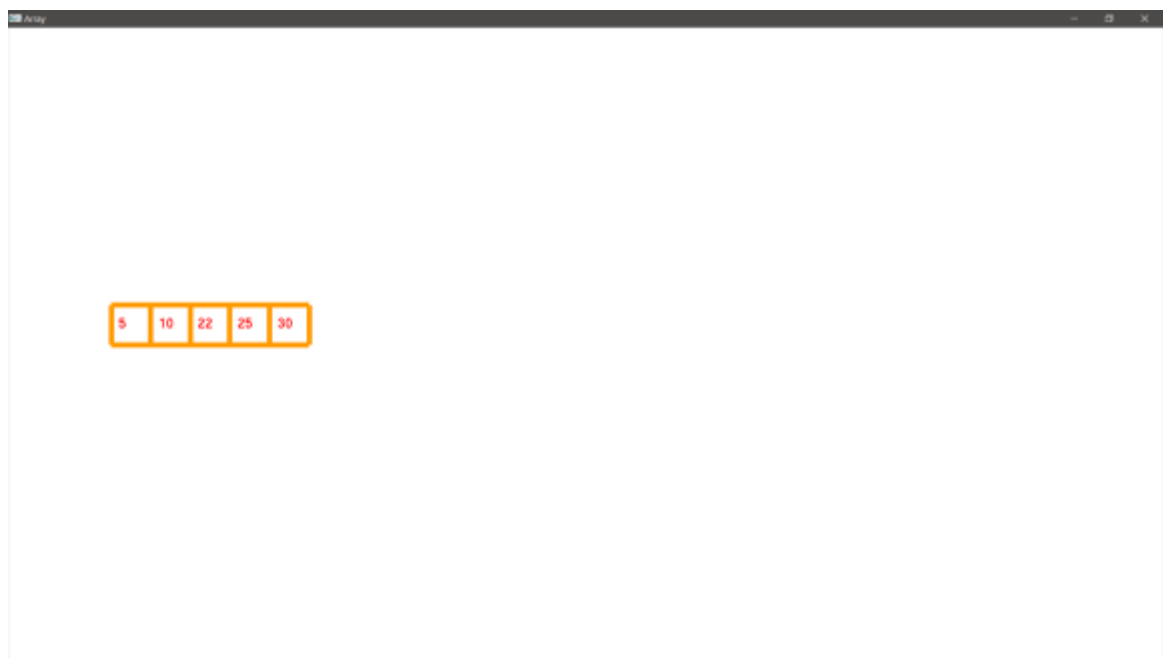


Figure 6.8: Array update

CONCLUSION

This mini project is intended to illustrate the solution for 'Array operations', a famous and basic operation. This project is shows illustration with a 3D view. This project can serve as a tool to understand the array operations in much simpler way. We have used many built-in functions which are user-friendly. We have tried to use many of the graphic concepts using OpenGL. There can be further enhancement in the project.

FUTURE ENHANCEMENT





This mini project is intended to illustrate the operations of an array in graphical way. This project supports basic functionalities such as insert, delete, search, sort and update. Currently limited number of data can only be inserted in the illustration and linear method is used for sorting. There can be further enhancement in the project like inserting more data into the array and also to use much simpler method to sort the data in the array.

REFERENCES

- 1) Donald Hearn & Pauline Baker: computer Graphics with OpenGL Version, 3rd Edition, Pearson Education, 2011.
- 2) [Online] <https://www.khronos.org/registry/OpenGL-Refpages/gl4/>
- 3) [Online] <https://www.youtube.com/watch?v=-BlgUU3-RFA>
- 4) [Online] <https://www.youtube.com/watch?v=ZDDZqhBspqI>

SIMULATION OF ARRAY OPERATIONS

PERSONAL PROFILE

	<p>Prof. Pradeep G. S received the B.E. degree in Information Science and Engineering from VCET Puttur College in the year 2010, and MTech (CSE) Canara college Engineering, Bantwal in 2013.</p> <p>His subjects of interest include Network security.</p> <p>Prof. Pradeep G S is pursuing his Ph.D.</p>
 <small>PhotoScanned by Google Photos</small>	<p>Name: Manish J USN: 4SU18CS039 Email: 4su18cs039@sdmit.in</p>
	<p>Name: Nithin Thomas USN: 4SU18CS052 Email: 4su18cs052@sdmit.in</p>
	<p>Name: Nivil Shibu USN: 4SU18CS053 Email: 4su18cs053@sdmit.in</p>