**Chapter 1**

**INTRODUCTION**

* 1. **INTRODUCTION TO COMPUTER GRAPHICS**

Computer Graphics is concerned with all aspects of producing pictures or images using a computer.

**Applications of Computer Graphics**

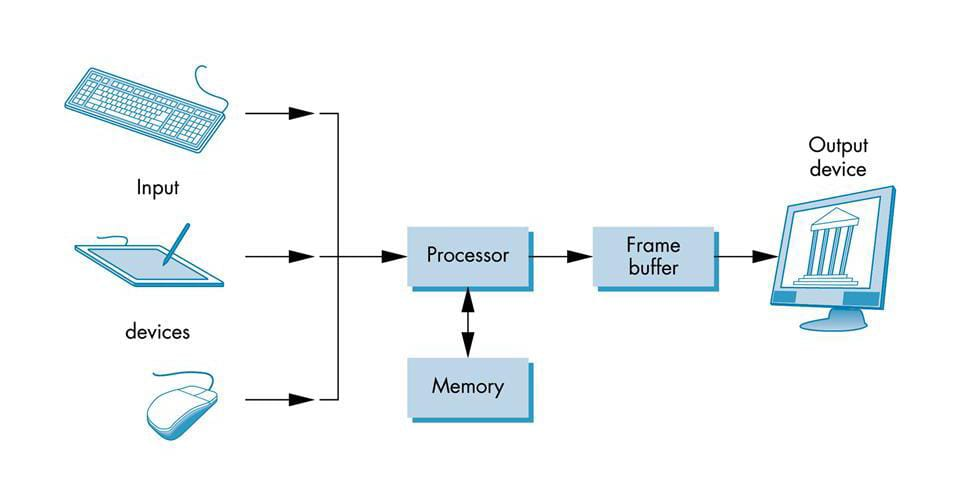
1. Display of Information
2. Design
3. Simulation and Animation
4. User Interfaces

**The Graphics Architecture Graphics Architecture can be made up of seven components:**

1. Display processors
2. Pipeline architectures
3. The graphics pipeline
4. Vertex processing
5. Clipping and primitive assembly
6. Rasterization
7. Fragment processing

**Components of Graphics Architecture and their working**

* **Input**: Graphics make use of various devices for data input. Most systems have a Keyboard and one or more additional devices specifically designed for interactive input. These include a mouse, trackball, space ball and joystick. Some other input devices used in particular applications are digitizers, dials, image scanners and voice system.
* **Devices**: Here device may be Input device or Output device. Any device that allows information from outside the computer to be communicated to the computer is considered as Input device. Any device which is used to send data from a computer to another device or user.
* **Mouse:** A computer mouse is a hand-held pointing device that detects two- dimensional motion relative to the surface. This motion is typically translated into motion of a pointer on a display, which allows a smooth control of the graphical user interface.
* **Processor:** A graphics processing unit is able to render images more quickly than a central processing unit because of its parallel processing architecture, which allows it to perform multiple calculations at the same time.
* **Memory:** The term memory identifies data storage that comes in the form of chips, and the word storage is used for memory that exists on tapes or disks. Every computer comes with a certain amount of physical memory referred as main memory.
* **Frame Buffer:** A frame buffer is a portion of RAM containing a bitmap that drives a video display. It is a memory buffer containing a complete frame of data. The information in buffer typically consists of color values for every pixel to be shown on the display.
* **Output Device:** Graphics hardware output devices are devices that generates computer graphics and allow them to be shown on a display.



**Figure 1.1:** Components of Graphics Architecture and their working.

* 1. **INTRODUCTION TO OPENGL**

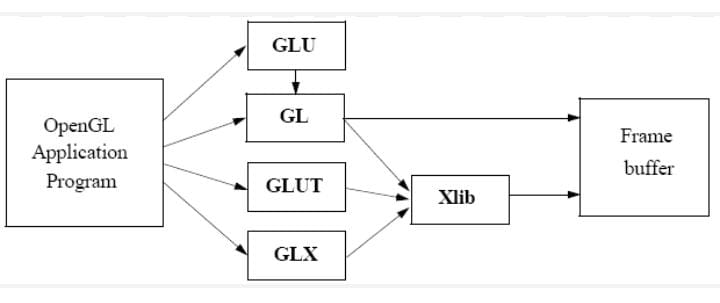
OpenGL is software used to implement computer graphics. The structure of OpenGL is similar to that of most modern APIs including Java 3D and DirectX. OpenGL is easy to learn, compared with other.

APIs are nevertheless powerful. It supports the simple 2D and 3D programs. It also supports the advanced rendering techniques. OpenGL API explains following components.

* Graphics functions
* Graphics pipeline and state machines
* The OpenGL interfaces

There are so many polygon types in OpenGL like triangles, quadrilaterals, strips and fans. There are 2 control functions, which will explain OpenGL through

* Interaction with window system
* Aspect ratio and view ports

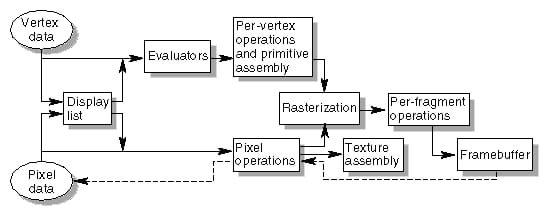


**Figure 1.2:** OpenGL Library Organization.

* OpenGL (Open Graphics Library) is the computer industry's standard application program interface (API) for defining 2-D and 3-D graphic images.
* The OpenGL Utility Library (GLU) was a computer graphics library for OpenGL. It consists of a number of functions that use the base OpenGL library to provide higher level drawing routines from the more primitive routines that OpenGL provides.
* GLUT is the OpenGL Uility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API, so you can write a single OpenGL program that works across all PC and workstation OS platforms.
* GL (Graphics Library): Library of 2-D,3-D drawing primitives and operations.
* GLX ("OpenGL Extension to the X Window System") is an extension to the X Window System core protocol providing an interface between OpenGL and the X Window System as well as extensions to OpenGL itself. It enables programs wishing to use OpenGL to do so within a window provided by the X Window System.
* Xlib is a library which serves as an interface between a programming language and the hardware of the graphics card and monitor. It abstracts details about the graphics card and monitor hardware.
* Xtk is a WebGL framework providing an easy-to-use API to visualize scientific data on the web.
* A frame buffer (frame buffer, or sometimes frame store) is a portion of RAM containing a bitmap that drives a video display. It is a memory buffer containing a complete frame of data. Modern video cards contain frame buffer circuitry in their cores. This circuitry converts an in-memory bitmap int0 a video signal that can be displayed on a computer monitor.
* Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 1.3, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do. The following diagram shows the assembly line approach, which OpenGL takes to process data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo

the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the frame buffer.

* **Display Lists:** All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.
* **Evaluators:** All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.



**Figure 1.3:** OpenGL Order of Operations.

* **Per-Vertex Operations:** For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.
* **Primitive Assembly:** Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped.
* **Pixel Operations:** While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.
* **Texture Assembly:** An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them.
* **Rasterization:** Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support anti aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.
* **Fragment Operations:** The first operation which may be encountered is texturing, where a Texel is generated from texture memory for each fragment and applied to the fragment. Then fog calculations may be applied, followed by the scissor test, the alpha test, the stencil test, and the depth-buffer test. Failing an enabled test may end the continued processing of a fragment's square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed. Finally, the thoroughly processed fragment is drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.
  1. **About Project**

Merge Sort is a sorting algorithm, which is commonly used in computer science. Merge Sort is a divide and conquer algorithm. It works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. So Merge Sort first divides the array into equal halves and then combines them in a sorted manner.

**Important Characteristics of Merge Sort:**

* Merge Sort is useful for sorting arrays and linked lists.
* Merge Sort is a stable sort which means that the same element in an array maintain their original positions with respect to each other.
* Overall time complexity of Merge sort is O(nlogn). It is more efficient as it is in worst case also the runtime is O(nlogn) The space complexity of Merge sort is O(n).
* This means that this algorithm takes a lot of space and may slower down operations for the last data sets.

**Chapter 2**

**REQUIREMENTS AND SPECIFICATIONS**

**2.1 Hardware Requirement**

* Processor : Intel Core i3+ /O
* Devices : Keyboard, Monitor
* RAM : 4GB or higher
* Hard disk : 40GB minimum

**2.2 Software Requirement**

* Operating system : Windows 10+ or Ubuntu.
* Compiler: g++ compiler.
* Application : gedit (Ubuntu).
* Languages : Open GL,C++. Libraries A built-in graphics library like glut and header files like <glut.h>, <stdlib.h>, <math.h> etc are used.

**Chapter 3**

**SYSTEM DESIGN**

Systems design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

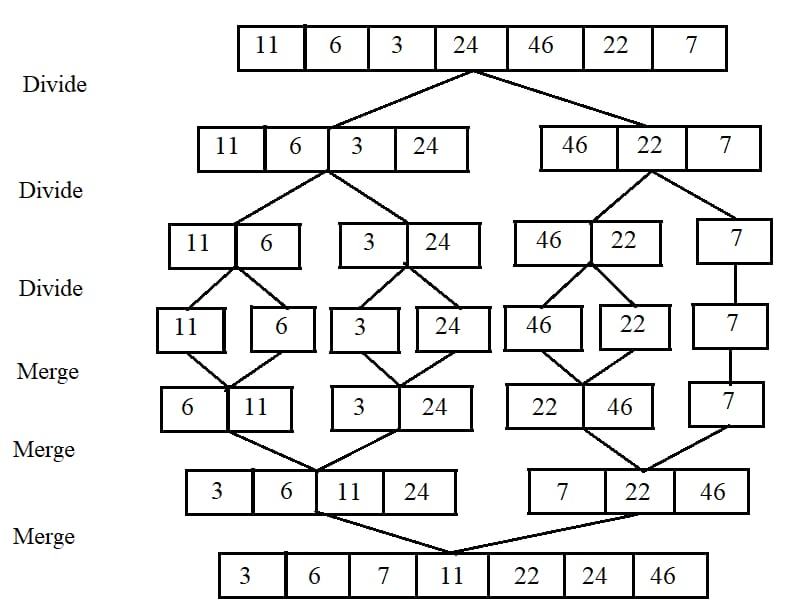
**3.1 User Inputs**

**Controls:**

* + When user runs the program in ubuntu a display page is appeared on screen.
  + User needs to press "ENTER" key from keyboard in order to go to next page.
  + User must enter the number of elements to sort, and the numbers to sort.
  + Then pressing the Enter key will give the visualization of the merge sort algorithm.

**3.2 Block Diagram**

Let's see the below example where we have five nodes that show the shortest path between the source and the destination.



**Figure 3.1:** Working of Merge sort Algorithm.

**Working**

* It divides the array into two halves. After that, it sorts the two arrays in a ascending order.
* It merges the two halves together and it easily becomes a complete single sorted array.
* Merge Sort mechanism divided the array first into two equal halves. Then it sorts the respective halves getting a sorted array at each end.
* Finally, the two halves are also equated whether the left index is greater than the right or vice versa and then the number is put into the array. In this way, the array is sorted.

**Chapter 4**

**IMPLEMENTATION**

**4.1 Pseudo Code**

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines. The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm.

**4.2 Main Function**

int main(int argc, char\*\* argv) {

// Read the number of elements in the array

std::cout << "Enter the number of elements: ";

std::cin >> n;

// Read the elements of the array

std::cout << "Enter the elements of the array:\n";

for (int i = 0; i < n; i++) {

std::cin >> array[i];

}

// Find the minimum and maximum values in the array

minVal = array[0];

maxVal = array[0];

for (int i = 1; i < n; i++) {

if (array[i] < minVal) {

minVal = array[i];

}

if (array[i] > maxVal) {

maxVal = array[i];

}

}

// Initialize OpenGL

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("Merge Sort");

init();

glutDisplayFunc(display); // Register the display function

// Perform merge sort

merge\_sort(array, 0, n - 1);

glutMainLoop();

return 0;

}

**4.3 Function for Merge Sort**

// Function to perform merge sort

void merge\_sort(int arr[], int low, int high) {

if (low < high) {

int mid = low + (high - low) / 2;

merge\_sort(arr, low, mid);

merge\_sort(arr, mid + 1, high);

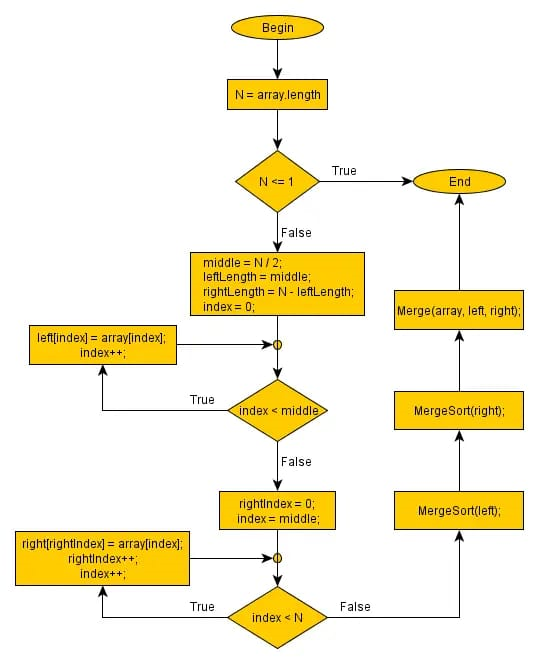
merge(arr, low, mid, high);

}

}

**4.4 Flow Chart**

A flowchart is a type of diagram that represents an algorithm, workflow or process. Flowchart can also be defined as a diagrammatic representation of an algorithm (step by step approach to solve a task). The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields



**Figure 4.1:** Flow Chart for implementation of Merge Sort Algorithm.

**4.5 Functions Used**

* **merge(int arr[], int low, int mid, int high):** This function performs the merge operation in the merge sort algorithm. It takes an array **arr** and three indices **low, mid**, and **high**. It merges two sorted subarrays: **arr[low..mid]** and **arr[mid+1..high],** into a single sorted subarray **arr[low..high].**
* **merge\_sort(int arr[], int low, int high):** This is the main function that implements the merge sort algorithm. It recursively divides the array into smaller subarrays until the base case is reached (when **low < high**). It then merges the subarrays using the **merge()** function.
* **display():** This function is responsible for displaying the array elements using OpenGL. It clears the color buffer, calls **draw\_stuff()** to draw additional visual elements, and then iterates through the array elements. It uses OpenGL functions to draw rectangles representing the elements and displays the numbers inside the rectangles.
* **init():** This function is used to initialize the OpenGL environment. It sets the clear color and specifies the orthographic projection.
* **print\_bitmap\_string(char\* s):** This function is used to print a string using bitmap characters in OpenGL. It takes a character pointer **s** and iterates through the characters, calling **glutBitmapCharacter** to draw each character.
* **draw\_stuff():** This function is responsible for drawing additional visual elements using OpenGL. It is called within the **display()** function and includes code to draw a text string on the screen.
* **main():** This is the entry point of the program. It reads the number of elements and the array elements from the user. Then it finds the minimum and maximum values in the array. It initializes the OpenGL environment, registers the **display()** function as the display callback, performs the merge sort algorithm on the array, and enters the OpenGL main loop using **glutMainLoop().**
* These functions work together to implement the merge sort algorithm and visualize the sorting process using OpenGL.

**Chapter 5**

**TESTING**

**5.1 Introduction to Testing**

Verification and validation are generic names given to checking processes, which that the software confirms to its specification and meets the demands of the user.

**Validation**

Are we building the right product? Validation involves checking that the program confirms to its specification.

**5.2 Stages in the Implementation of Testing**

* **Unit Testing**

Each individual unit is tested for the program correctness. These individual components will be tested to ensure that they operate correctly.

* **System Testing**

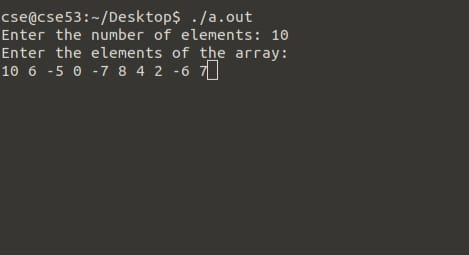
The sub systems are integrated to make up the entire system. The errors that result from unanticipated interaction between sub system and system components are removed.

* **User Acceptance Testing**

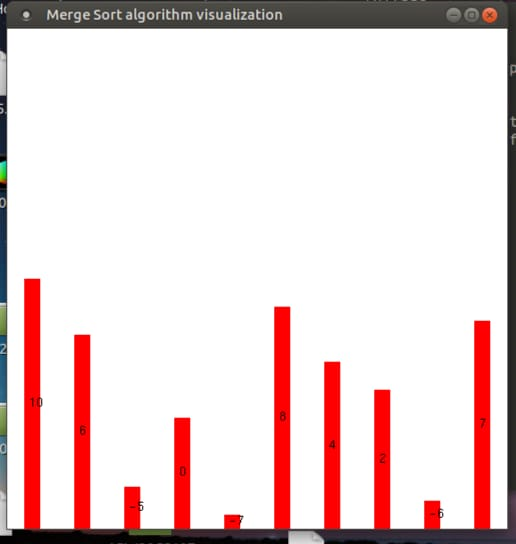
This is the final stage in the testing process before the system is tested for operational use. Any requirements problem or requirement definition problem revealed from acceptance testing are considering and made error file.

**Chapter 6**

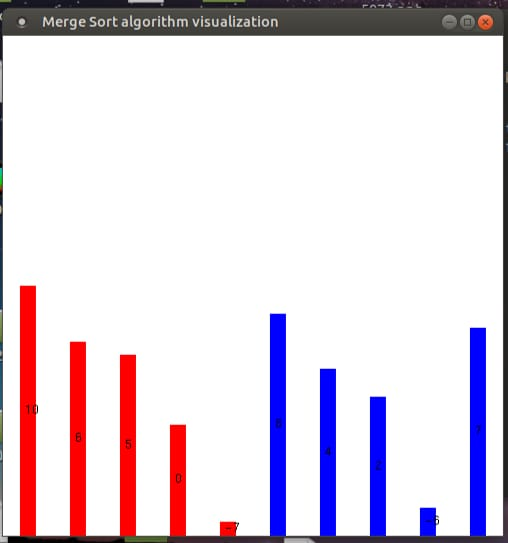
**SNAPSHOTS**



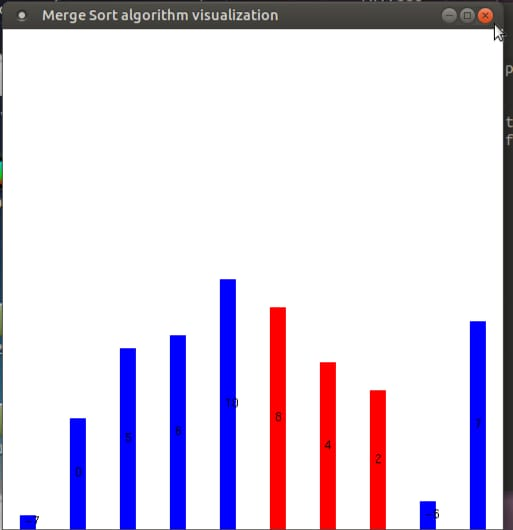
**Figure 6.1:** Entering the array elements.

****

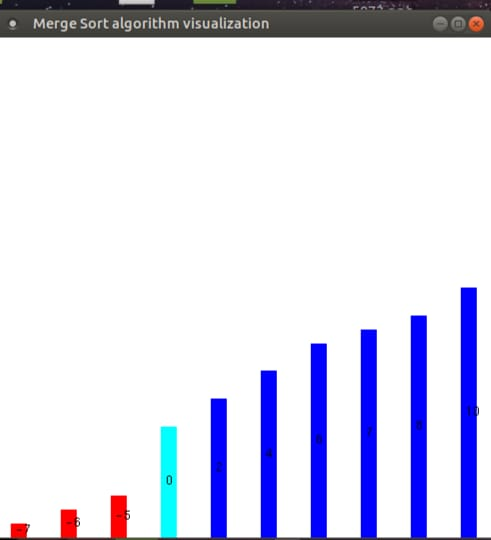
**Figure 6.2:** Randomly placed array elements.



**Figure 6.3:** Two halves of sorted array.

****

**Figure 6.4 :** Sorting of array by dividing them.

****

**Figure 6.5:** Fully Sorted array element.