**CHAPTER 1**

**INTRODUCTION**

* 1. **Overview of computer graphics**

Computer Graphics become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage so it is not surprising to find the use of CG so widespread.

Although early application in engineering & science had to rely on expensive and cumbersome equipment, advances in computer technology have made interactive computer graphics a practical tool. Today people find Computer Graphics in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education and training. Computer graphics as generalized tool for drawing and creating pictures and simulate the real world situations within a small computer window.

**1.2 History**

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeing. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmel and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

**1.3 Applications of computer graphics**

Nowadays Computer Graphics used in almost all the areas ranges from science, engineering, medicine, business, industry, government, art, entertainment, education and training.

**1.3.1 CG in the field of CAD**

Computer Aided Design methods are routinely used in the design of buildings,

Automobiles, aircraft, watercraft, spacecraft computers, textiles and many other applications.

**1.3.2 CG in presentation Graphics**

Another major application area presentation graphics used to produce illustrations for reports or generate slides. Presentation graphics is commonly used to summarize financial, statistical, mathematical, scientific data for research reports and other types of reports.2D and 3D bar chart to illustrate some mathematical or statistical report.

**1.3.3 CG in computer Art**

CG methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods including special purpose hardware, artist’s paintbrush program (lumena), other pain packages, desktop packages, mathematics packages, animation packages that provide facility for designing object motion. Ex: cartoons design is an example of computer art which uses CG.

**1.3.4 Entertainment**

Computer graphics methods are now commonly used in making motion pictures, music, videos, games and sounds. Sometimes graphics objects are combined with the actors and live scenes.

**1.3.5 Education and Training**

Computer generated models of physical financial, economic system is often as education aids. For some training application special systems are designed. Ex: specialized system is simulator for practice sessions or training of ship captain, aircraft pilots and traffic control.

**1.3.6 Image Processing**

Although the methods used in CG image processing overlap, the 2 areas are concerned with fundamentally different operations. In CG a computer is used to create picture. Image processing on the other hand applies techniques to modify existing pictures such as photo scans, TV scans.

**1.4 User interface**

It is common for software packages to provide a graphical interface. A major component of a graphical interface is a window manager that allows a user to display multiple window area.Interface also displays menus,icons for fast selection and processing.

* 1. Problem statement

Learning computer graphics as a part of academics and implementing it as a project on OpenGL made us to implement on sorting techniques i.e. sorting in OpenGL which included all the basic library functions.

* 1. Objectives of the project
* The main objective this project is to show how the “Sorting” technique works with the help of built-in graphics library functions.
* Learning how to build a simulation of sorting.
* Illustrating about the keyboard and mouse interaction.
* To provide sorted data to the user.

1.7Organization of the report

This section deals with the Introduction and organization of the project report. Chapter 2 discusses the basic concepts OpenGL. Chapter 3 discusses the basic concept and working principle. Chapter 4 gives information about the design and implementation Chapter 5 include results and snapshot and. Chapter 6 gives the conclusion and future enhancement of the project.

.

**CHAPTER 2**

**INTRODUCTION TO OPENGL**

**2.1 Introduction**

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

Most of our application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows). The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with letters glu.

To interface with the window system and to get input from external devices into our programs, developer needs at least one more library. For each major window system there is a system-specific library that provides the “glu” between the window system and OpenGL. For the X window system, this library is called GLX, for windows, it is wgl, and for the Macintosh, it is agl. Rather than using a different library for each system, here it uses a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

OpenGL application Program

GLU

GL

GLUT

GLX

Xlib, Xtk

Frame Buffer

**Fig 2.1: Library organization of OpenGL**

**2.2 OpenGL command syntax**

OpenGL commands use the prefix **gl**and initial capital letters for each word making up the command name. Similarly, OpenGL defined constants begin with GL\_, use all capital letters, and use underscores to separate words (like GL\_TRUE).

Some extraneous letters are appended to some command names (for example, the 3f in glColor3f() and glVertex3f()). It's true that the Color part of the command name glColor3f() is enough to define the command as one that sets the current color. However, more than one such command has been defined so as to use different types of arguments. In particular, the 3 part of the suffix indicates that three arguments are given; another version of the Color command takes four arguments. The f part of the suffix indicates that the arguments are floating-point numbers. Having different formats allows OpenGL to accept the user's data.

**2.3 Pixel operation**

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. If pixel data is read from theframe buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

**2.4 Texture assembly**

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory.

**2.5 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 initializing 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

Processor

Rasterizer

Clipper and

Primitive Assembler

Vertex Processor

vertices Pixel

**Figure 2.2 Block diagram showing Rasterization**

**2.6Immediate mode and 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

#### Transforming to Window Coordinates

Before clip coordinates can be converted to window *coordinates*, they are normalized by dividing by the value of w to yield *normalized device coordinates*. After that, the viewport transformation applied to these normalized coordinates produces window coordinates. One can control the viewport, which determines the area of the on-screen window that displays an image, with glDepthRange() and glViewport().

#### Matrix Transformations

Vertices and normals are transformed by the modelview and projection matrices before they're used to produce an image in the frame buffer. You can use commands such as glMatrixMode(), glMultMatrix(), glRotate(), glTranslate(), and glScale() to compose the desired transformations, or you can directly specify matrices with glLoadMatrix() and glLoadIdentity(). Use glPushMatrix() and glPopMatrix() to save and restore modelview and projection matrices on their respective stacks.

* The basic model for OpenGL command interpretation is immediate mode, in which a command is executed as soon as the server receives it; vertex processing, for example, may begin even before specification of the primitive of which it is a part has been completed. Immediate mode execution is well-suited to interactive applications in which primitives and modes are constantly altered. In OpenGL, the fine-grained control provided by immediate mode is taken as far as possible: even individual lighting parameters (the diffuse reflectance color of a material, for instance) and texture images are set with individual commands that have immediate effect. While immediate mode provides flexibility, its use can be inefficient if unchanging parameters or objects must be re-specified. To accommodate such situations, OpenGL provides display lists. A display list encapsulates a sequence of OpenGL commands (all but a handful of OpenGL commands may be placed in a display list), and is stored on the server. The display list is given a numeric name by the application when it is specified; the application need only name the display list to cause the server to effectively execute all the commands contained within the list. This mechanism provides a straightforward, effective means for an application to transmit a group ofcommands to the server just once even when those same commands must be executed many times.

**2.6.1 Display List Optimization**

Accumulating commands into a group for repeated execution presents possibilities for optimization. Consider, for example, specifying a texture image. Texture images are often large, requiring a large, and therefore possibly slow, data transfer from client to server (or from the server to its graphics subsystem) whenever the image is re-specified. For this reason, some graphics subsystems are equipped with sufficient storage to hold several texture images simultaneously. If the texture image definition is placed in a display list, then the server may be able to load that image just once when it is specified. When the display list is invoked (or re-invoked), the server simply indicates to the graphics subsystem that it should use the texture image already present in its memory, thus avoiding the overhead of re-specifying the entire image. Examples like this one indicate that display list optimization is required to achieve the best performance. In the case of texture image loading, the server is expected to recognize that a display list contains texture image information and to use that information appropriately. This expectation places a burden on the OpenGL implementer to make sure that special display list cases are treated as efficiently as possible. It also places a burden on the application writer to know to use display lists in cases where doing so could improve performance. Another possibility would have been to introduce special commands for functions that can be poor performers in immediate mode. But such specialization would clutter the API and blur the clear distinction between immediate mode and display lists.

**2.7 Advantages of using OpenGL**

* Industry standard: An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

Stable: OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes.

* Backward compatibility requirements ensure that existing applications do not become obsolete.
* Reliable and portable: All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.
* Evolving: Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

**CHAPTER 3**

**BASIC CONCEPT AND WORKING PRINCIPLE**

Sorting technique is a brute force approach in which a given list of un-orderable items is been rearranged in a nondecreasing order. Dozens of algorithms have been developed in solving the important problem but here we will discuss the concept of Bubble sort and Merge sort technique.

**3.1 Selection Sort**

In selection sort we first find the smallest item and we exchange it with the first item. Obtain the second smallest in the list and exchange it with the second element and so on. Finally, all the items will be arranged in ascending order. Since, the next least item is selected and exchanged appropriately so that elements are finally sorted , this technique is called Selection Sort.A pictorial representation of sorting is been shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A[0]=45 | A[1]=20 | A[2]=40 | A[3]=5 | A[4]=15 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A[0]=5 | A[1]=20 | A[2]=40 | A[3]=45 | A[4]=15 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A[0]=5 | A[1]=15 | A[2]=40 | A[3]=45 | A[4]=20 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A[0]=5 | A[1]=15 | A[2]=20 | A[3]=45 | A[4]=40 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A[0]=5 | A[1]=15 | A[2]=20 | A[3]=40 | A[4]=45 |

***Fig 3.1 : implementation of selection sort.***

Time complexity of selection sort is:

f (n)=

So, time complexity of selection sort =θ (

**3.2 Merge Sort**

In merge sort a given element is divided into two parts. The left part of the array as well as the right part of the array is sorted recursively. Later the sorted right parts are finally merged into single sorted vector. Suppose we have two sorted vectors A and B. Assume m elements are in vector A and n elements in vector B. To obtain a sorted vector, compare ith item of vector A with jth vector of vector b and copy a lesser item into kth position of resultant vector C. that is A[i] is less than B[j], copy the item A[i] to the resultant vector C[k] and update the index variables I, k, l. This process is repeated for 0<i<m and 0<=j<n. When i and j exceeds its limits, the element remaining in the other vector can be copied with vector C. The merging of two sorted arrays can be done by initializing two pointers to a point to the first element of the arrays being merged. Then the elements pointed are being compared and the smaller of them is added to a new array being constructed; after tht, the index of the smaller element is incremented to a point to its immediate successors in the array it was copied from. This operation is continued until one of the two given arrays is exhausted, and then the remaining elements of the other array are copied to the end of the new array.

A pictorial representation of sorting is been shown below:

1 2 3 4 5 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **77** | **42** | **35** | **12** | **101** | **5** |

1 2 3 4 5 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **5** | **12** | **35** | **42** | **77** | **101** |

***fig 3.2: Showing the final sorting using Merge Sort***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **77** | **42** | **35** | **12** | **101** | **5** |

|  |  |  |
| --- | --- | --- |
| **77** | **42** | **35** |

|  |  |  |
| --- | --- | --- |
| **12** | **101** | **5** |

|  |
| --- |
| **77** |

|  |
| --- |
| **42** |

|  |
| --- |
| **5** |

|  |
| --- |
| **35** |

|  |
| --- |
| **12** |

|  |
| --- |
| **101** |

|  |  |  |
| --- | --- | --- |
| **35** | **42** | **77** |

|  |  |  |
| --- | --- | --- |
| **5** | **12** | **101** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **5** | **12** | **35** | **42** | **77** | **101** |

***fig 3.3: “Merge sort operation” of all elements***

Actually, it is not necessary to copy the second half of array A to the auxiliary array B. It can remain where it is in array A. In this way, only half as much auxiliary memory space is required, and also only half as much time to copy array elements to B. Furthermore, if all elements of the first half have been copied back to A, the remaining elements of the second half need not be moved anymore since they are already at their proper places.

The straightforward version of function merge requires at most 2nsteps (n steps for copying the sequence to the intermediate array B, and at most n steps for copying it back to array A). The [time complexity](http://www.iti.fh-flensburg.de/lang/algorithmen/asympen.htm) of merge sort is therefore

T (n) = 2n + 2 T (n/2)   and

T (1) = 0

The solution of this recursion yields

T (n)  2nlog (n)    O (nlog (n))

Thus, the merge sort algorithm is optimal, since the [lower bound](http://www.iti.fh-flensburg.de/lang/algorithmen/asympen.htm)for the sorting problem of Ω (n log (n)) is attained.

In the more efficient variant, function merge requires at most 1.5n steps (n/2 steps for copying the first half of the sequence to the intermediate array B, n/2 steps for copying it back to array A, and at most n/2 steps for processing the second half). These yields a running time of merge sort of at most 1.5nlog (n) steps.

Algorithm merge sort has a time complexity of Θ (nlog (n)) which is optimal. A drawback of merge sort is that it needs an additional space of Θ (n) for the temporary array B.

There are different possibilities to implement function merge. The most efficient of these is variant B. It requires only half as much additional space, it is faster than the other variants, and it is stable.

**3.3 Quick sort:**

The quick sort is based on the principal of “Divide and Conquer”. The quick sort works on large set of data. Unlike mergesort , which divides them its input’s elements according to their position in the array , quicksort divides them according to their value. Specifically , it rearranges elements of a given array A[0….*n*-*1*] to achieve its partition , a situation where all the elements before some positions are smaller than or equal to A[s] and all the elements after position s are greater than or equal to A[s]:

A [0]…..A[s-1] A[s] A[s+1]…..A[n-1]

All are ≤ A [s-1] all are ≥ A [s]

Obviously, after a partition has been achieved , A[s] will be in its final position in the sorted array , and we can continue sorting the two subarray of elements preceding and following A[s] independently.

**Implementation of quicksort:**

**0 1 2 3 4 5 6 7**

**5** 1 9 8 2 4

**5** 3 1 8 2 7

**5** 3 1 8 2 7

**5** 3 1 4 9 7

**5** 3 1 4 9 7

**5** 3 1 4 9 7

2 3 1 4 **5** 8 9 7

**2** 1

**2** 4

**2** 4

**2** 4

1 **2**3 4

1

**0 1 2 3 4 5 6 7**

**3**

4

**8**

**8**

**8**

7 **8** 9

7

9

**0 1 2 3 4 5 6 7**

***Fig3.4: Implementation of quicksort.***

Time complexity of quicksort is T (n) = θ (n)

**CHAPTER 4**

**DESIGN AND IMPLEMENTATION**

**4.1 Design**

The whole program has been implemented in C language. The bottom line of the design is Translation function, mouse interaction and keyboard interaction are included and some minor functions are used to print the text on the screen. In this project, ellipse is the major part and it is created by using simple polygon. By using the polygon, one can easily create the ellipse.

**4.2 Implementation**

The OpenGL provides very powerful translation facilities which relive the programmers by allowing them to concentrate on their jobs rather than focusing on how to implement these operations

**4.2.1 Built in functions**

* **void glutInitDisplayMode(unsigned int mode)**

*mode* – specifies the display mode. Use the *mode* to specify the colour mode

and the number of type of buffers.

* **void glutInitWindowSize(int width, int height)**

Windows created by glutCreateWindow will be requested to be created with the current initial window position and size. The initial value of the initial window position GLUT state is -1 and -1. If either the X or Y component to the initial window position is negative, the actual window position is left to the window system to determine. The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero.

.

* **void glutWindowPosition(int x, int y)**

*x* -Window X location in pixels.

*y* -Window Y location in pixels.

glutWindowPositionset the initial window position and size respectively.

* **Int glutCreateWindow(char \*name);**

glutCreateWindow creates a top-level window. The name will be provided to the window system as the window’s name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly created window. Each created window has a unique associated OpenGL context. State changes to a window’s associated OpenGL context can be done immediately after the window is created.

* **void glutDisplayFunc(void (\*func)(void));**

glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback.

* **void glutMouseFunc(void (\*func)(int button, intstate,int x, int y));**

*func--* The new mouse callback function.

glutMouseFunc sets the mouse callback for the *current window*.

* **void glutKeyboardFunc(void (\*func)(char key,intwidth,int height));**

glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character.

* **void glViewport(intx,inty,GLsizeiwidth,GLsizei height);**

Specifies a width\*height viewport in pixels whose lower-left corner is at (x,y)

measured from the origin of the window.

* **void glutMainLoop(void);**

glutMainLoop enters the GLUT event processing loop.

* **glPushMatrix() and glPopMatrix()**

Pushes to and pops from the matrix stack corresponding to the current matrix

mode.

* **void glFlush()**

forces any buffered openGL commands to execute.

**,**

* **void glClear(GL\_COLOR\_BUFFER\_BIT)**

To make the screen solid and black.

* **gluquadricObj\* gluNewQuadric()**

Returns a pointer to a new quadric object.

**4.2.2 User defined function**

* **void graphics() :**This is the routine which is used to displayfront page on the screen. It ispassed as an argument to glutDisplayFunc (). In this function we Enable GL\_LIGHTING using glEnable(GL\_LIGHTING); And also initialize corresponding parameters for GL\_ LIGHTING.
* **void init2()**

To initialize OpenGL. It sets orthogonal viewing volume.

* **void drawstring(GLfloat,GLfloat,char\*) :**This function displays given input strings on the output window. We use void glutBitmapCharacter(void \*font, int character); this function print the character on window.
* **void delay ()**

This function is used to set delay.

* **void draw\_tab1()**

This function draws rectangles for the sorted elements of selection sort.

* **void selection()**

This function sorts the input array using selection sort technique.

* **void myDisplay1()**

This function displays the sorted list of elements for selection sort. This function call selection() function to run selection sort technique. Result is display in the box in the Display window by calling draw\_table1() function.

* **void Merge()**

This function divides the array of n elements into two arrays of size n/2.

* **void Mergesort()**

This function merges the sorted array elements.

* **void quick()**

This function partitions the input array based on first and last element.

* **void sort\_menu()**

This function compares the user’s choice with the case and performs respective actions .If user select selection sort this function call selection sort function using switch() keyword and same for Merge sort and Quick sort.

**4.3 Hardware and Software requirements**

**Hardware requirements:**

* Processor : Intel Pentium-4, Intel Pentium Dual Core, Intel Pentium Core2duo
* Hard disc : 40GB and above
* RAM : 256MB and above
* Monitor : CRT or TFT

**Software requirements:**

* Operating system: Windows 2000, Windows XP, Windows VISTA. Windows 7
* Application: Visual studio 6.0.

**4.4 Flow Chart**

The following diagram shows the flow chart of the project. It shows the flow of control throughout the program and the sort. This is very helpful for the user to know the usage of the simulation. By seeing this flow chart user can easily analyze the implementation.

Right click on mouse

SelectiEnter the array size

on

Enter the number of elements

ESelection sort

Case 1

Merge

ize

Sel Enter the number of elements

Enter the array size

ments

rt

Case 1

Merge

Enter the number of elements

Enter the array size

Merge sort

Case 2

Case 3

Quick

Enter the number of elements

Enter the array size

Quick sort

default: Quit

**Fig.4.1 Flow chart of Sorting**

The above flow chart gives the procedure for using this Sorting simulation.

**CHAPTER 5**

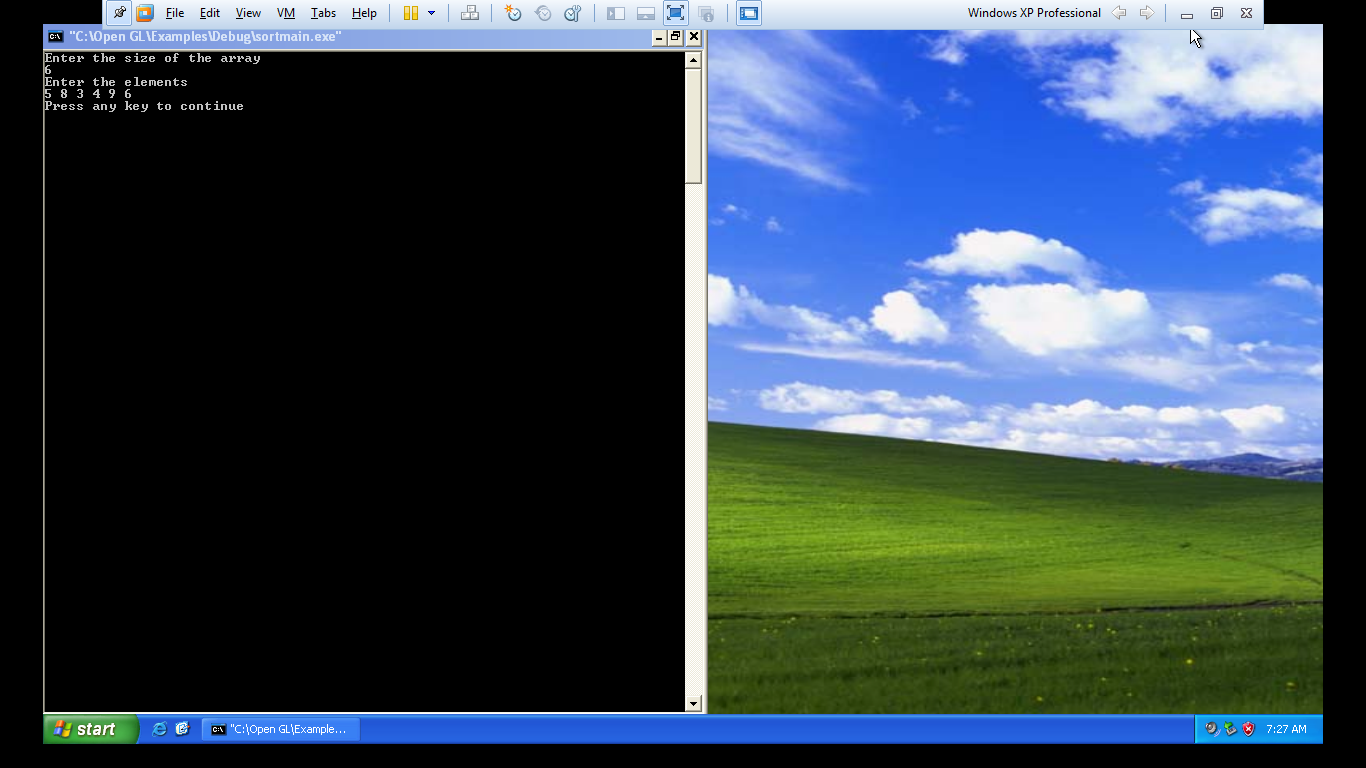
**RESULTS AND SNAPSHOTS**



***Fig 5.1 Snapshot showing front sheet of project***



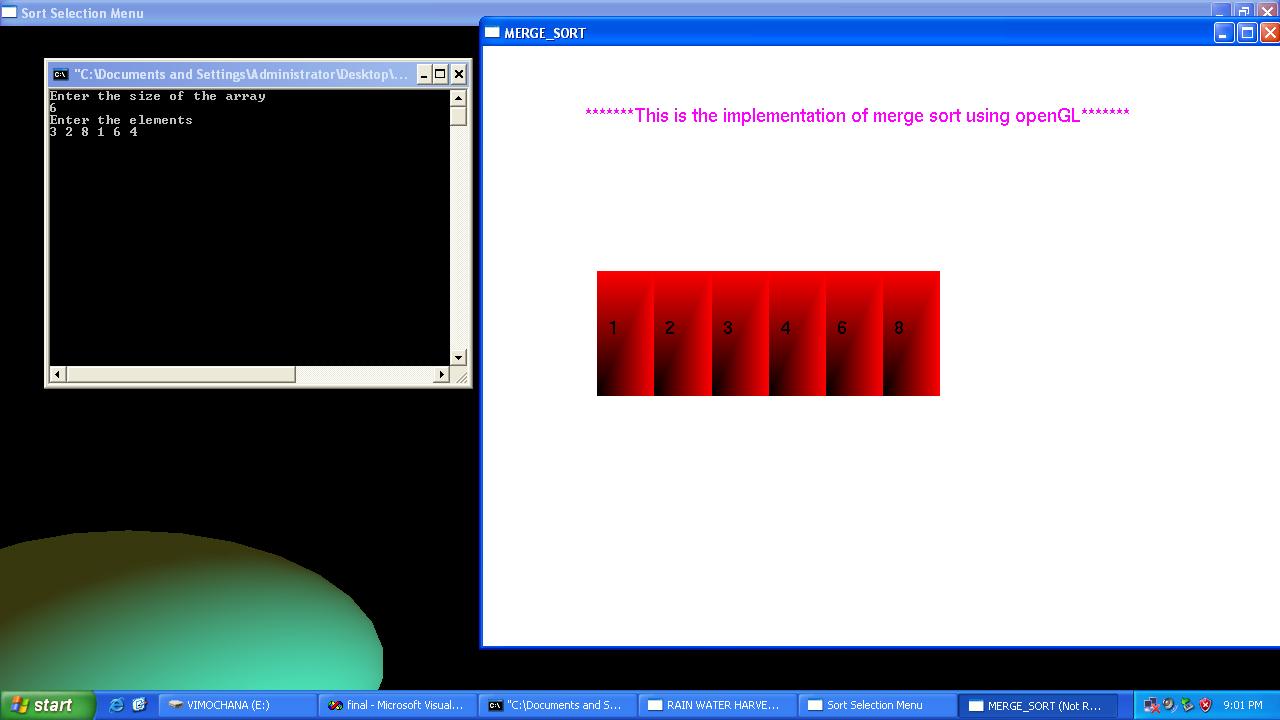
***Fig 5.2 Snapshots showing sorting techniques available.***



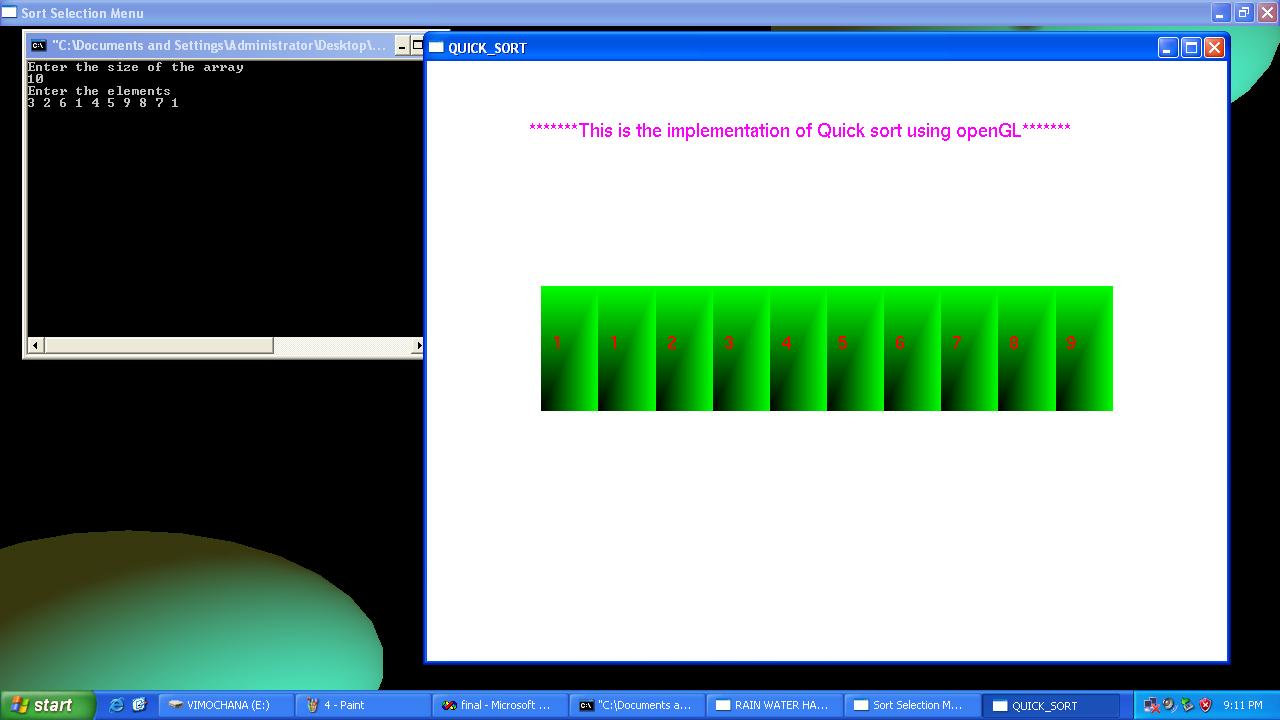
***Fig 5.3 Snapshot of input array***



***Fig 5.4: Snapshot showing the sorting technique in Selection Sort***



***Fig 5.4: Snapshot showing the sorting technique in Merge Sort***



***Fig 5.4: Snapshot showing the sorting technique in Quicksort***

**CHAPTER 6**

**CONCLUSION & FUTURE ENHANCEMENT**

**Conclusion:**

* The Selection Sort , Merge sort and Quick sort techniques depend upon Brute Force and Divide and conquer rule respectively, both are required to sort unordered elements but in their own unique way.
* Here we can graphically understand better how the sorting techniques work.
* We can visualize both the algorithm and thus figure out which one is better.

**Future Enhancement:**

These algorithms can be used in the future to solve the problems like Triomino Puzzle, Nuts and Bolts Program etc.

By using some more OpenGL functions we can make User-interface of this program simpler and more user friendly will certainly help beginners in using this program more easily.

It can be made still more attractive by using 3D implementation and make user interactive.

**REFERENCES**

* + Interactive Computer Graphics A Top-Down Approach with OpenGL – Edward Angel, 5th edition, Addison-Wesley, 2008.
  + Introduction to the Design And Analysis of Algorithm – AnanyLevetin, 3rd edition, 2007.
  + Computer Graphics using OpenGL- F.S.Hill,Jr. 2nd edition, Pearson education, 2001.
  + [www.aw.com/cssupport](http://www.aw.com/cssupport)
  + [www.opengl.org](http://www.opengl.org)