VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI-590018

COMPUTER GRAPHICS AND IMAGE PROCESSING (21CSL66)

Mini Project Report

on

SCAN Algorithm Visualization

Submitted in partial fulfillment of the requirements for the VI semester

Computer Science and Engineering of Visvesvaraya Technological University, Belagavi

Submitted by:

Navaneeth N *1RN21CS098*
Rahul G Athreyas *1RN21CS118*

Under the Guidance of:
Dr. Sudhamani M J
Associate Professor

Department of Computer Science and Engineering
RNS Institute of Technology
Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098

2024

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project work entitled **SCAN Algorithm visualisation** has been successfully carried out by **Navaneeth N and Rahul G Athreyas** bearing USN **1RN21CS098 and 1RN21CS118** respecitively, bonafide student of **RNS Institute of Technology** in partial fulfillment of the requirements for the 6th semester **Computer Science and Engineering of Visvesvaraya Technological University"**, Belagavi, during academic year 2024. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the COMPUTER GRAPHICS AND IMAGE PROCESSING laboratory requirements of 6th semester BE, CSE.

Signature of the Guide	Signature of the HoD	Signature of the Principal
Dr. Sudhamani M J	**Dr. Kiran P**	**Dr. Ramesh Babu H S**
Associate Professor	Professor & Head	Principal
Dept. of CSE	Dept. of CSE	

External Viva:

Name of the Examiners	Signature with Date
1.	
2.	

Acknowledgement

The successful completion of any achievement is not solely dependent on individual efforts but also on the guidance, encouragement, and cooperation of intellectuals, elders, and friends. We would like to take this opportunity to express our heartfelt gratitude to all those who have contributed to the successful execution of this project.

First and foremost, we extend our profound thanks to **Sri. Satish R Shetty**, Managing Trustee of R N Shetty Trust and Chairman of RNS Group of Institutions, and **Sri. Karan S Shetty** , CEO of RNS Group of Institutions, Bengaluru, for providing a conducive environment that facilitated the successful completion of this project.

We would also like to express our sincere appreciation to our esteemed Director, **Dr. M K Venkatesha**, for providing us with the necessary facilities and support throughout the duration of this work.

Our heartfelt thanks go to our respected Principal, **Dr. Ramesh Babu H S**, for his unwavering support, guidance, and encouragement that played a vital role in the completion of this project.

We would like to extend our wholehearted gratitude to our HOD, **Dr. Kiran P**, Professor, and Head of the Department of Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice, which greatly contributed to the success of this endeavor.

A special word of thanks is due to our project guide,**Dr. Guide Name**, Associate Professor in the Department of CSE, RNSIT, Bangalore, for her exceptional guidance, constant encouragement, and unwavering assistance throughout the project.

We would also like to express our sincere appreciation to all the teaching and non-teaching staff of the Department of Computer Science & Engineering, RNSIT, for their consistent support and encouragement.

Once again, we express our deepest gratitude to everyone involved, as their support and cooperation were instrumental in the successful completion of this project.

Abstract

The SCAN disk scheduling algorithm, also known as the elevator algorithm, optimizes disk read/write operations by moving the read-write head across the disk in a linear fashion, servicing requests in one direction before reversing. In my computer graphics project, I visualized this algorithm using OpenGL and C++. The visualization features a plot illustrating the read-write head's movement from the initial position to subsequent seek positions. The disk is represented as concentric circles, each track comprising 20 sectors. As the head moves, the track containing the current seek point lights up, providing a dynamic and educational display of the SCAN algorithm in action.

Contents

Acknowledgement i

Abstract ii

List of Figures v

List of Tables vi

1 Introduction 1

1.1 Overview of Computer Graphics 1

1.2 History of Computer Graphics 2

1.3 Applications of Computer Graphics 3

1.3.1 Display of information 3

1.3.2 Design 3

1.3.3 Simulation and Animation 3

1.3.4 User interfaces 4

1.4 Overview of Image Processing 4

1.4.1 Image Acquisition 4

1.4.2 Image Enhancement 4

1.4.3 Image Restoration 5

1.4.4 Color Image Processing 5

1.4.5 Image Segmentation 6

1.4.6 Image Compression 6

1.4.7 Image Representation and Description 6

1.4.8 Image Recognition 6

1.5 Applications 7

1.5.1 Medical Imaging 7

1.5.2 Remote Sensing 7

1.5.3 Computer Vision 7
1.5.4 Industrial Inspection 7
1.5.5 Security and Surveillance 7
1.6 Conclusion 7

2 OpenGL **8**
2.1 OpenGL Libraries 8
2.2 OpenGL Contributions 9
2.3 Limitations 9

3 Resource Requirements **10**
3.1 Hardware Requirements 10
3.2 Software Requirements 10

4 System Design **11**

5 Implementation **13**

6 Testing **29**

7 **30**
7.1 Results & Snapshots 30

8 Conclusion & Future Enhancements **33**
8.1 Conclusion 33
8.2 Future Enhancements 33

References **34**

List of Figures

7.1 Left path traversal - 1 30
7.2 Left path traversal - 2 30
7.3 Right path traversal - 1 31
7.4 Right path traversal - 2 31
7.5 Total seek operations - left direction 32
7.6 Total seek operations - right direction 32

List of Tables

3.1 Hardware Requirements 10
3.2 Software Requirements 10

6.1 Test Case Validation 29

Chapter 1

Introduction

1.1 Overview of Computer Graphics

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. Typically, the term computer graphics refers to several different things:

- The representation and manipulation of image data by a computer.
- The various technologies used to create and manipulate images.
- The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Today, computers and computer-generated images touch many aspects of daily life. Computer images is found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, thesis, and other presentation material.Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics.

As technology has improved, 3D computer graphics have become more common. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis

is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic component.

1.2 History of Computer Graphics

In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software.Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location.

In 1961 another student at MIT, Steve Russell, created the first video game, E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963. During 1970s, the first major advance in 3D computer graphics was created at University of Utah by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology.

3D graphics became more popular in the 1990s in gaming, multimedia and animation. In 1996, Quake, one of the first fully 3D games, was released. In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas worldwide. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modeling software.

1.3 Applications of Computer Graphics

The applications of computer graphics can be divided into four major areas:

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

1.3.1 Display of information

Computer graphics has enabled architects, researchers and designers to pictorially interpret the vast quantity of data. Cartographers have developed maps to display the celestial and geographical information. Medical imaging technologies like Computerized Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound, Positron Emission Tomography (PET) and many others make use of computer graphics.

1.3.2 Design

Professions such as engineering and architecture are concerned with design. They start with a set of specification; seek cost-effective solutions that satisfy the specification. Designing is an iterative process. Designer generates a possible design, tests it and then uses the results as the basis for exploring other solutions. The use of interactive graphical tools in Computer Aided Design (CAD) pervades the fields including architecture, mechanical engineering, and the design of very-large-scale integrated (VLSI) circuits and creation of characters for animation.

1.3.3 Simulation and Animation

Once the graphics system evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. Graphical flight simulators have proved to increase the safety and to reduce the training expenses. The field of virtual reality (VR) has opened many new horizons. A human viewer can be equipped with a display headset that allow him/her to see the images with left eye and right eye which gives the effect of stereoscopic vision. This has further led to motion pictures and interactive video games.

1.3.4 User interfaces

Computer graphics has led to the creation of graphical user interfaces (GUI) using which even naive users are able to interact with a computer. Interaction with the computer has been dominated by a visual paradigm that includes windows, icons, menus and a pointing device such as mouse. Millions of people are internet users; they access the internet through the graphical network browsers such as Microsoft internet explorer and Mozilla Firefox.

1.4 Overview of Image Processing

Image processing is a method to perform operations on an image to enhance it or extract useful information. It is a type of signal processing where the input is an image and the output can be either an image or a set of characteristics/features related to the image. It involves various steps like image acquisition, enhancement, restoration, segmentation, and more. Following subsections describe the basic steps in in Image Processing

1.4.1 Image Acquisition

Image acquisition is the first step in image processing. It involves capturing an image using a sensor (such as a camera) and converting it into a digital form that can be processed by a computer. The quality of the acquired image significantly affects the subsequent processing steps. Examples include capturing a photograph using a digital camera or scanning a document using a scanner.

1.4.2 Image Enhancement

Image enhancement involves improving the visual appearance of an image or converting the image to a form better suited for analysis by a human or machine. Techniques used for image enhancement include:

- **Contrast Adjustment**: Modifying the contrast of an image to make it more suitable for visual interpretation.
- **Histogram Equalization**: Improving the contrast of an image by redistributing the intensity values.
- **Noise Reduction**: Removing noise from an image using filters such as Gaussian, median, or adaptive filters.

1.4.3 Image Restoration

Image restoration aims to recover an image that has been degraded by known factors. This step involves reversing the degradation to retrieve the original image. Common techniques include:

- **De-blurring**: Removing blurriness caused by motion or out-of-focus lenses.
- **Noise Filtering**: Reducing noise while preserving important details and edges in the image.
- **Inverse Filtering**: Applying the inverse of the degradation function to restore the image.

1.4.4 Color Image Processing

Color image processing involves the manipulation of color images and is a crucial area in image processing as it adds an extra dimension to the visual content. The primary goals are color correction, color enhancement, and color space transformation.

Color Models

Different color models represent colors in various ways, each suitable for different applications. Common color models include:

- **RGB (Red, Green, Blue)**: Used in electronic displays and cameras.
- **CMY(K) (Cyan, Magenta, Yellow, Black)**: Used in color printing.
- **HSV (Hue, Saturation, Value)**: Useful for color analysis and manipulation.

Color Space Transformations

Transforming an image from one color space to another is essential for various processing techniques. For instance:

- **RGB to Grayscale**: Converts a color image to grayscale by removing hue and saturation information while retaining luminance.
- **RGB to HSV**: Helps in separating color information (hue) from intensity information (value).

Color Correction and Enhancement

Improving the color quality of an image involves techniques like:

- **White Balance Adjustment**: Corrects color casts due to different lighting conditions.

- **Histogram Equalization**: Enhances contrast by adjusting the intensity distribution.

1.4.5 Image Segmentation

Dividing an image into meaningful segments or regions is crucial for further analysis and interpretation. Techniques include:

- **Thresholding**: Simple and effective for binary segmentation based on pixel intensity.
- **Edge Detection**: Identifies object boundaries using operators like Sobel, Canny, and Prewitt.
- **Clustering**: Groups pixels with similar attributes using algorithms like K-means and Mean Shift.

1.4.6 Image Compression

Reducing the size of an image file without significantly degrading its quality is vital for storage and transmission. There are two main types of compression:

- **Lossless Compression**: Reduces file size without losing any data (e.g., PNG, GIF).
- **Lossy Compression**: Reduces file size by sacrificing some quality (e.g., JPEG).

1.4.7 Image Representation and Description

Transforming image data into a form suitable for computer processing involves various techniques:

- **Shape Representation**: Uses boundaries and regions to describe object shapes.
- **Boundary Descriptors**: Includes curvature, Fourier descriptors, and chain codes.
- **Regional Descriptors**: Characterizes objects based on their region properties like area, centroid, and texture.

1.4.8 Image Recognition

Identifying and classifying objects within an image is essential for many applications. Techniques include:

- **Pattern Recognition**: Utilizes statistical methods to recognize patterns in data.
- **Neural Networks**: Employs deep learning for highly accurate image recognition.

- **Machine Learning Algorithms**: Uses algorithms like Support Vector Machines (SVM) and Random Forest for classification tasks.

1.5 Applications

1.5.1 Medical Imaging

Medical imaging involves enhancing and analyzing medical images for diagnostic and treatment purposes. Techniques such as MRI, CT scans, and X-rays are processed to improve clarity and assist in accurate diagnosis.

1.5.2 Remote Sensing

Remote sensing involves processing satellite or aerial images to monitor and analyze earth's surface. Applications include environmental monitoring, agricultural assessment, and urban planning.

1.5.3 Computer Vision

Computer vision enables machines to interpret and make decisions based on visual data. Applications include self-driving cars, facial recognition, and automated inspection systems.

1.5.4 Industrial Inspection

Industrial inspection uses image processing for quality control and fault detection in manufacturing processes. Techniques include checking for defects, verifying dimensions, and ensuring product quality.

1.5.5 Security and Surveillance

Security and surveillance applications involve using image processing for tasks such as face recognition, motion detection, and behavior analysis to enhance security measures.

1.6 Conclusion

Image processing is integral to modern technology, driving advancements in numerous fields. With continuous advancements in computational power and algorithms, the capabilities and applications of image processing continue to expand, offering innovative solutions across various industries.

Chapter 2

OpenGL

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data 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. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

2.1 OpenGL Libraries

Computer Graphics are created using OpenGL, which became a widely accepted standard software system for developing graphics applications. As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer. OpenGL stands for 'open graphics library' graphics library is a collection of API's (Applications Programming Interface). Graphics library functions are:

- GL library (OpenGL in windows) – Main functions for windows.

- GLU (OpenGL utility library) - Creating and viewing objects.
- GLUT (OpenGL utility toolkit)- Functions that help in creating interface of windows

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. These libraries are included in the application program using preprocessor directives OpenGL User Interface Library (GLUI) is a C++ user interface library based on the OpenGL Utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window and operating system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management. The OpenGL Utility Library (GLU) is a computer graphics library. 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. It is usually distributed with the base OpenGL package.

2.2 OpenGL Contributions

It is very popular in the video games development industry where it competes with Direct3D (on Microsoft Windows).OpenGL is also used in CAD, virtual reality, and scientific visualization programs.OpenGL is very portable. It will run for nearly every platform in existence, and it will run well. It even runs on Windows NT 4.0 etc. The reason OpenGL runs for so many platforms is because of its Open Standard. OpenGL has a wide range of features, both in its core and through extensions. Its extension feature allows it to stay immediately current with new hardware features, despite the mess it can cause.

2.3 Limitations

- OpenGL is case sensitive.
- Line Color, Filled Faces and Fill Color not supported.
- Shadow plane is not supported.

Chapter 3

Resource Requirements

3.1 Hardware Requirements

The Hardware requirements are very minimal and the program can be run on most of the machines.Table 3.1 gives details of hardware requirements.

Table 3.1: Hardware Requirements

Processor	Intel Core i3 processor
Processor Speed	1.70 GHz
RAM	4 GB
Storage Space	40 GB
Monitor Resolution	1024*768 or 1336*768 or 1280*1024

3.2 Software Requirements

The software requirements are description of features and functionalities of the system.Table 3.2 gives details of software requirements.

Table 3.2: Software Requirements

Operating System	Windows 8.1
IDE	Microsoft Visual Studio with C++ 2022
OpenGL libraries	glut.h,glu32.lib,opengl32.lib,glut32.lib glu32.dll,glut32.dll,opengl32.dll.

Chapter 4

System Design

The description of all the functions used in the program is given below:

- **glutInitDisplayMode(GLUT_SINGLE — GLUT_RGB) :**
  - **Purpose:** Sets the initial display mode for the window.
  - **Parameters:**
    - **GLUT_SINGLE:** Specifies single-buffered mode. Single buffering means drawing directly to the screen.
    - **GLUT_RGB:** Specifies the use of an RGB color model.
- **glutDisplayFunc(display) :**
  - **Purpose:** Registers the display callback function that GLUT calls whenever the window needs to be redrawn.
  - **Parameters:**
    - **display:** The function to call whenever the window needs to be redrawn.
- **glutMouseFunc(mouse) :**
  - **Purpose:** Registers the mouse callback function that GLUT calls when a mouse event occurs.
  - **Parameters:**
    - **mouse:** The function to call when a mouse button is pressed or released.

- **glutTimerFunc(int milliseconds, timerFunc, int value) :**
  - **Purpose:** Registers a timer callback function to be triggered after a specified number of milliseconds.
  - **Parameters:**
    - **milliseconds:** The number of milliseconds to wait before calling the timer function.
    - **timerFunc:** The function to call when the timer expires.
    - **value:** An integer value that will be passed to the timer function when it is called.

Chapter 5

Implementation

```
#include<gl/glut.h>
#include <iostream>
#include<stdio.h>
#include<string.h>
#include<math.h>
//globals
int req_seq[] = { 176, 79, 34, 60, 100, 11, 41, 114 };
int head = 90;
int track[] = { 1,0,0,0,0,0,0,0,0,0 };
int points[] = { 0,20,40,60,80,100,120,140,160,180,200,220 };
int point[10][2];
int k = 0;
int left = 0;
int in3 = 0;
int currentLineIndex = 1;
const int totalLines = 10;
float thickness = 3.0f;
int delay = 1500;

// ----------- Incremental Line Drawing Algorithm -----------

//set the pixel values
void setPixel(int x, int y) {
   glColor3f(0.7, 1, 1);
   glBegin(GL_POINTS);
```

```
   glVertex2i(x, y);
   glEnd();
}
//implementation of incremental line drawing algorithm
void plotLineIncremental(int x0, int y0, int x1, int y1, float thickness) {
   int dx = abs(x1 - x0);
   int dy = abs(y1 - y0);
   int x = x0;
   int y = y0;
   int sx = (x0 < x1) ? 1 : -1;
   int sy = (y0 < y1) ? 1 : -1;

   if (dx > dy) {
      int p = 2 * dy - dx;
      int inc1 = 2 * dy;
      int inc2 = 2 * (dy - dx);

      for (int i = -thickness / 2; i <= thickness / 2; i++) {
         x = x0;
         y = y0 + i;
         int offset = i;

         for (int j = x0; j != x1; j += sx) {
            setPixel(x, y);
            if (p >= 0) {
               y += sy;
               p += inc2;
            }
            else {
               p += inc1;
            }
            x += sx;
         }
         setPixel(x, y);
      }
   }
```

```
    else {
        int p = 2 * dx - dy;
        int inc1 = 2 * dx;
        int inc2 = 2 * (dx - dy);

        for (int i = -thickness / 2; i <= thickness / 2; i++) {
            x = x0 + i;
            y = y0;
            int offset = i;

            for (int j = y0; j != y1; j += sy) {
                setPixel(x, y);
                if (p >= 0) {
                    x += sx;
                    p += inc2;
                }
                else {
                    p += inc1;
                }
                y += sy;
            }
            setPixel(x, y);
        }
    }
}
//draw dotted line from seek point till etched seek points below
void draw_dotted_lines(int x, int y) {
    glColor3f(1, 1, 1);
    glEnable(GL_LINE_STIPPLE);
    glLineStipple(1, 0xCCCC);
    glLineWidth(0.5f);
    glBegin(GL_LINES);
    glVertex2f(x, y);
    glVertex2f(x, 36);
    glEnd();
    glDisable(GL_LINE_STIPPLE);
```

```
   glFlush();
}
//setup the order of seek points to be serviced
void plot() {
   int init_y = 435, init_x = (head * 2) + 100;
   glColor3f(0.8, 0.2, 0.2);

   // Sort the seek points
   for (int i = 0; i < 8; i++) {
      for (int j = i + 1; j < 8; j++) {
         if (req_seq[i] > req_seq[j]) {
            int t = req_seq[i];
            req_seq[i] = req_seq[j];
            req_seq[j] = t;
         }
      }
   }
   //left direction
   if (left == 0) {
      int head_pos = -1;
      for (int i = 0; i < 8; i++) {
         if (req_seq[i] < head) {
            head_pos = i;
         }
         else {
            break;
         }
      }
      init_x = (head * 2) + 100;
      point[0][0] = init_x;
      point[0][1] = init_y;
      int k = 1;
      for (int i = head_pos; i >= 0; i--) {
         int x = (req_seq[i] * 2) + 100;
         int y = init_y - (k * 40);
         point[k][0] = x;
```

```
        point[k++][1] = y;
    }
    point[k][0] = 100;
    point[k++][1] = init_y - (k * 40);
    for (int i = head_pos + 1; i < 8; i++) {
        int x = (req_seq[i] * 2) + 100;
        int y = init_y - (k * 40);
        point[k][0] = x;
        point[k++][1] = y;
    }
    currentLineIndex = 1;
}
//right direction
else {
    int head_pos = -1;
    for (int i = 0; i < 8; i++) {
        if (req_seq[i] >= head) {
            head_pos = i;
            break;
        }
    }
    init_x = (head * 2) + 100;
    point[0][0] = init_x;
    point[0][1] = init_y;

    int k = 1;
    for (int i = head_pos; i < 8; i++) {
        int x =(req_seq[i] * 2) + 100;
        int y = init_y - (k * 40);
        point[k][0] = x;
        point[k++][1] = y;
    }
    point[k][0] = 540;
    point[k++][1] = init_y - (k * 40);
    for (int i = head_pos - 1; i >= 0; i--) {
        int x = (req_seq[i] * 2) + 100;
```

```
            int y = init_y - (k * 40);
            point[k][0] = x;
            point[k++][1] = y;
        }
        currentLineIndex = 1;
    }
    glutTimerFunc(0, timerFunc, 0);
}
// ----------- String print functions -----------

//display with large font size
void output_large(int x, int y, char* string) {
    glColor3f(1.0, 0.647, 0.0);
    int len, i;
    glRasterPos2f(x, y);
    len = (int)strlen(string);
    for (i = 0; i < len; i++)
    {
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
    }
    glFlush();
}
//display with small font size
void output_small(int x, int y, char* string) {
    glColor3f(1,1,1);
    int len, i;
    glRasterPos2f(x, y);
    len = (int)strlen(string);

    for (i = 0; i < len; i++)
    {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, string[i]);
    }
    glFlush();
}
//screen2 content display
```

```
void output_content(float x, float y, void* font, const char* string) {
   glColor3f(1, 1, 1);
   const char* c;
   glRasterPos2f(x, y);
   for (c = string; *c != '\0'; c++) {
      glutBitmapCharacter(font, *c);
   }
   glFlush();
}
//display the points on graph
void output_number(int x, int y) {
   glColor3f(1, 1, 1);
   for (int i = 0; i < sizeof(points) / sizeof(points[0]); ++i) {
      int num = points[i];
      char str[5];
      int result = sprintf_s(str, sizeof(str), "%d", num);
      if (result < 0) {
         printf("Buffer error occurred\n");
      }
      else {
         if (num < 99)
            glRasterPos2f(x + (i * 40) - 11, y);
         else
            glRasterPos2f(x + (i * 40) - 15, y);
         int len = (int)strlen(str);
         for (int j = 0; j < len; j++) {
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, str[j]);
         }
      }
   }
   glFlush();
}
//display disk numbers
void output_disk(int x, int y) {
   glColor3f(0,0,0);
   int k = 0;
```

```
    for (int i = 20; i <= 220;i+=20 ) {
        int num = i;
        int x1 = x;
        int y1 = y + (k * 21);
        k++;
        char str[5];
        int result = sprintf_s(str, sizeof(str), "%d", num);
        if (result < 0) {
            printf("Buffer error occurred\n");
        }
        else {
            glRasterPos2f(x1,y1);
            int len = (int)strlen(str);
            for (int j = 0; j < len; j++) {
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, str[j]);
            }
        }
    }
    glFlush();
}
//display the seek points below
void output_down_number(int x, int y) {
    glColor3f(1, 1, 1);
    for (int i = 0; i < sizeof(req_seq) / sizeof(req_seq[0]); ++i) {
        int num = req_seq[i];
        char str[5];
        int result = sprintf_s(str, sizeof(str), "%d", num);
        if (result < 0) {
            printf("Buffer error occurred\n");
        }
        else {
            int adjusted_x = x + (num * 2);
            if (num < 99)
                glRasterPos2f(adjusted_x-5, y);
            else
                glRasterPos2f(adjusted_x - 10, y);
```

```
        int len = (int)strlen(str);
        for (int j = 0; j < len; j++) {
           glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10, str[j]);
        }
     }
   }
   glFlush();
}
// ----------- Drawing functions -----------

//underline functionality
void draw_line(int x, int y, int x1, int y1, float thickness) {
   glColor3f(1, 1, 1);
   glLineWidth(thickness);
   glBegin(GL_LINES);
   glVertex2f(x, y);
   glVertex2f(x1, y1);
   glEnd();
   glFlush();
}
//draw the disc
void draw_disk() {
   int r = 240;
   int x = 790, y = 260;
   for (int j = 0; j < 11; j++) {
      r = r - 20;
      k++;
      if (track[j]== 0)
         glColor3f(0.7, 0.7, 0.7);
      else
         glColor3f(0, 0.5019607843137255, 0.5019607843137255);
      glBegin(GL_POLYGON);
      for (float i = 0; i < 360; i += 0.01) {
         float x1 = r * cos(i) + x;
         float y1 = r * sin(i) + y;
```

```
            glVertex2f(x1, y1);
        }
        glEnd();
    }
    r = 220;
    glColor3f(0, 0, 0);
    for (int j = 1; j <= 11; j++) {
        r = r - 20;
        glBegin(GL_POINTS);
        for (float i = 0; i < 360; i += 0.1) {
            float x1 = r * cos(i) + x;
            float y1 = r * sin(i) + y;
            glVertex2f(x1, y1);
        }
        glEnd();
    }
}
//point the seek sequence points
void draw_circle(int x, int y) {
    int r = 4;
    if (x == 100 || x==540) {
        glColor3f(0, 0, 1);
    }
    else {
        glColor3f(1, 0, 0);
    }
    glBegin(GL_POLYGON);
    for (float theta = 0; theta < 360; theta++) {
        float x1 = r * cos(theta)+x;
        float y1 = r * sin(theta)+y;
        glVertex2f(x1, y1);
    }
    glEnd();
    glFlush();
}
//the graph polygon
```

```
void draw(float p[4][2]) {
   glColor3f(0, 1, 0);
   glLineWidth(3.0f);
   glColor3f(0, 0.5019607843137255, 0.5019607843137255);
   glBegin(GL_POLYGON);
   glVertex2fv(p[0]);
   glVertex2fv(p[1]);
   glVertex2fv(p[2]);
   glVertex2fv(p[3]);
   glEnd();
   glFlush();
}
//etch the points on graph
void marks(int x, int y) {
   int up = y - 5;
   int down = y + 5;
   for (int i = 1; i <= 10; i++) {
      glColor3f(1, 0, 0);
      int x1 = x - (i * 40);
      glBegin(GL_LINES);
      glVertex2f(x1, up);
      glVertex2f(x1, down);
      glEnd();
      glFlush();
   }
}
// ----------- screen implemetations -----------

void screen1() {
   in3 = 0;
   glutTimerFunc(0,NULL,0);
   char main[] = "SCAN Algorithm";
   char below[] = "Project by : ";
   char name1[] = "Navaneeth N";
   char name2[] = "Rahul G Athreyas";
   glClearColor(0, 0, 0, 0);
```

```
    glColor3f(1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    output_large(450, 350, main);
    output_large(800, 100, below);
    output_small(800, 70, name1);
    output_small(800, 45, name2);

}
void screen2() {
    in3 = 0;
    glutTimerFunc(0, NULL, 0);
    char head[] = "SCAN Algorithm";
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);

    output_content(50, 570, GLUT_BITMAP_HELVETICA_18, "Step 1: Initialize the scan
        line.");
    output_content(50, 540, GLUT_BITMAP_HELVETICA_18, "Step 2: Find intersections.");
    output_content(50, 510, GLUT_BITMAP_HELVETICA_18, "Step 3: Sort intersections.");
    output_content(50, 480, GLUT_BITMAP_HELVETICA_18, "Step 4: Fill pixels between
        pairs of intersections.");
    output_content(50, 450, GLUT_BITMAP_HELVETICA_18, "Step 5: Repeat for all scan
        lines.");
    output_content(70, 380, GLUT_BITMAP_HELVETICA_18, "In the SCAN Disk Scheduling
        Algorithm, the head starts from one end of the disk and moves towards the");
    output_content(50, 350, GLUT_BITMAP_HELVETICA_18, "other end, servicing requests
        in between one by one and reaching the other end. Then the direction of the");
    output_content(50, 320, GLUT_BITMAP_HELVETICA_18, "head is reversed and the
        process continues as head continuously scans back and forth to access the
        disk.");
    output_content(50, 290, GLUT_BITMAP_HELVETICA_18, "So, this algorithm works as
        an elevator and is hence also known as the elevator algorithm. As a result,");
    output_content(50, 260, GLUT_BITMAP_HELVETICA_18, "the requests at the midrange
        are serviced more and those arriving behind the disk arm will have to wait.");
```

```
   output_large(450, 680, head);
}
void screen3() {
   in3 = 1;
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   char head[] = "SCAN Algorithm";
   char problem[] = "Problem :";
   char req[] = "Request sequence = [ 176, 79, 34, 60, 92, 11, 41, 114 ]";
   char headpos[] = "Initial head position = 90";
   char trackpos[] = "Track:";
   output_large(450, 680, head);
   output_large(50, 600, problem);
   output_small(75, 550, req);
   output_small(75, 530, headpos);
   output_large(760, 520, trackpos);

   draw_circle(480, 435);
   float points[4][2] = { {100,35},{100,475},{540,475},{540,35} };
   draw(points);
   marks(540, 475);
   output_number(100, 490);
   output_down_number(100, 20);
   draw_disk();
   output_disk(785, 260);
   plot();
}
void screen4() {
   in3 = 0;
   glutTimerFunc(0, NULL, 0);
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   char head[] = "SCAN Algorithm";
   output_large(450, 680, head);

   char problem[] = "Answer :";
```

```
    output_large(50, 600, problem);
    int seek = 0;
    for (int i = 1; i < 10; i++) {
        int s;
        if ((point[i][0] / 2) > (point[i - 1][0] / 2)) {
            s = (point[i][0] / 2) - (point[i - 1][0] / 2);
        }
        else
            s = (point[i-1][0] / 2) - (point[i][0] / 2);
        seek += s;
    }

    char answer[] = "The total number of seek operations = ";
    output_small(75, 550, answer);
    char str[5];
    int result = sprintf_s(str, sizeof(str), "%d", seek);
    if (result < 0) {
        printf("Buffer error occurred\n");
    }
    else {
        glColor3f(1.0, 0.647, 0.0);
        glRasterPos2f(395, 550);
        int len = (int)strlen(str);
        for (int j = 0; j < len; j++) {
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, str[j]);
        }
    }
    glFlush();
}
// ----------- events -----------

void display() {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    screen1();
    glFlush();
```

```
}
void timerFunc(int value) {
   if (in3 == 1) {
      if (currentLineIndex < totalLines) {
         glColor3f(0.8, 0.2, 0.2);
         plotLineIncremental(point[currentLineIndex - 1][0], point[currentLineIndex
             - 1][1], point[currentLineIndex][0], point[currentLineIndex][1],
             thickness);
         if (point[currentLineIndex][0] != 100 && point[currentLineIndex][0] != 540)
             {
            draw_dotted_lines(point[currentLineIndex][0],
                point[currentLineIndex][1]);
            draw_circle(point[currentLineIndex-1][0], point[currentLineIndex-1][1]);
            for (int i = 0; i < 11; i++)
               track[i] = 0;
            int seek_point = (point[currentLineIndex][0] - 100) / 2;
            int index = seek_point / 20;
            printf("%d at %d \n ", seek_point, index);
            printf("------------\n");
            track[10-index] = 1;
            draw_disk();
            output_disk(785, 260);

         }
         currentLineIndex++;
         glutTimerFunc(delay, timerFunc, 0);
         draw_circle(point[currentLineIndex - 1][0], point[currentLineIndex - 1][1]);
         output_disk(785, 260);
      }
   }
}
void keyboard(unsigned char key, int x, int y) {
   if (key == '1')
      screen1();
   else if (key == '2')
      screen2();
```

```
   else if (key == '3')
      screen3();
   else
      screen4();
}
void mouse(int b, int s, int x, int y) {
   if (b == GLUT_LEFT_BUTTON && s == GLUT_DOWN && in3 == 1) {
      left = 0;
      screen3();
   }
   if (b == GLUT_RIGHT_BUTTON && s == GLUT_DOWN && in3 == 1) {
      left = 1;
      screen3();
   }
}
void myinit() {
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, 1040, 0, 720);
   glMatrixMode(GL_MODELVIEW);
}
void main() {
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(1040, 720);
   glutInitWindowPosition(0, 0);
   glutCreateWindow("SCAN Algorithm");
   myinit();
   glutDisplayFunc(display);
   glutMouseFunc(mouse);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
}
```

Chapter 6

Testing

In unit testing, individual program modules are tested separately to identify errors in code and logic. This approach ensures each module functions correctly on its own and maintains data integrity. Various routines were validated by passing inputs and verifying outputs. Table 6.1 details the test case validation.

Table 6.1: Test Case Validation

Test Case No.	**Metric**	**Description**	**Observation**
1	Animation Effect	Bars move either left or right depending on the sorting algorithm	Results obtained as expected.
2	Keyboard Function	Navigates to different pages of the project	Results obtained as expected.
3	Plotting Function	Plots the read-write head to corresponding sector points in scan order	Results obtained as expected.
4	Mouse Function	Reverses the direction of the read-write head movement - left on left-click and right on right-click	Results obtained as expected.
5	Calculation Function	Calculates the total seek operations as per the direction of the read-write head	Results obtained as expected.

Chapter 7

7.1 Results & Snapshots

The direction of read-write head is left

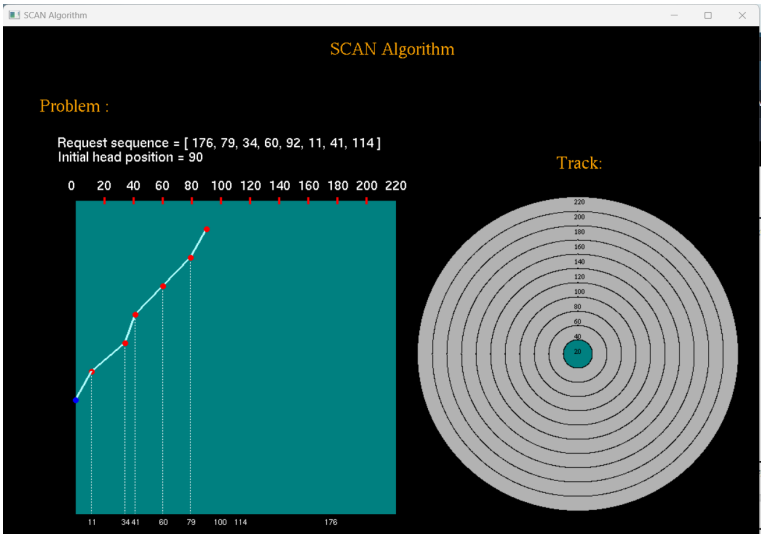

Figure 7.1: Left path traversal - 1

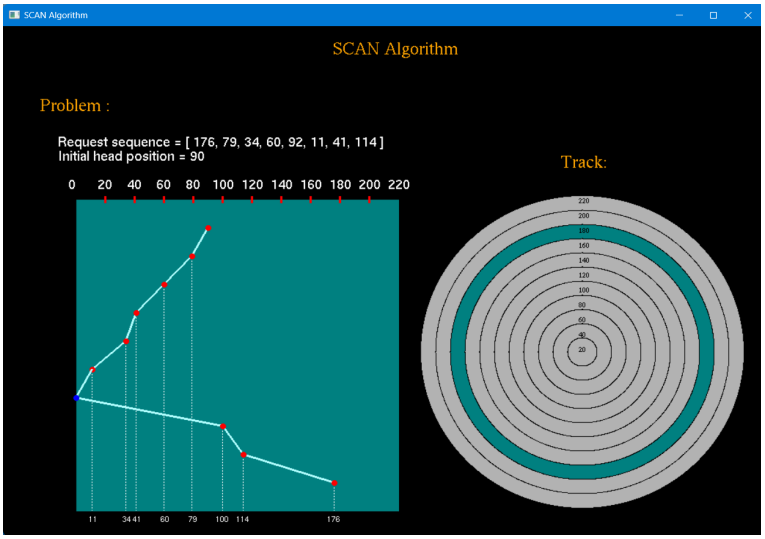

Figure 7.2: Left path traversal - 2

The direction of read-write head is right

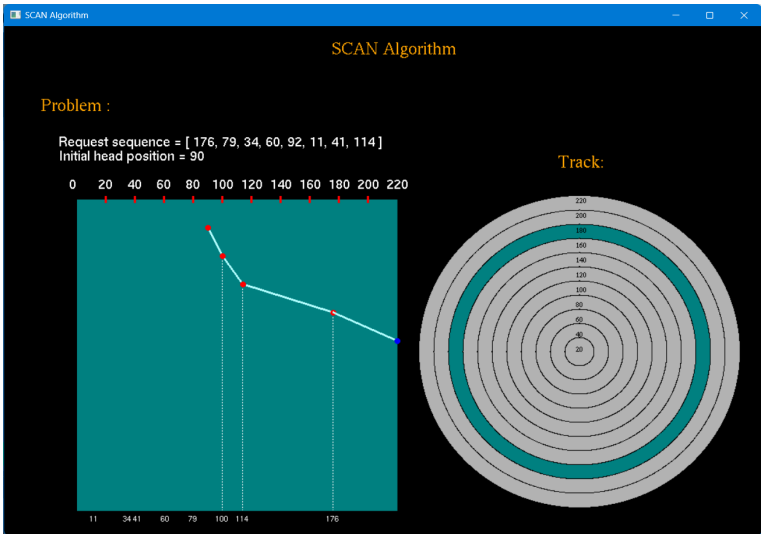

Figure 7.3: Right path traversal - 1

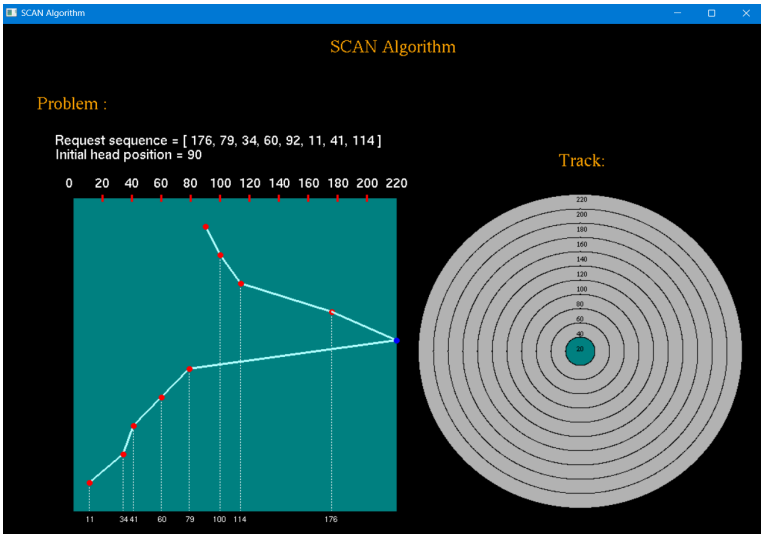

Figure 7.4: Right path traversal - 2

The resulting number of seek operations

Figure 7.5: Total seek operations - left direction

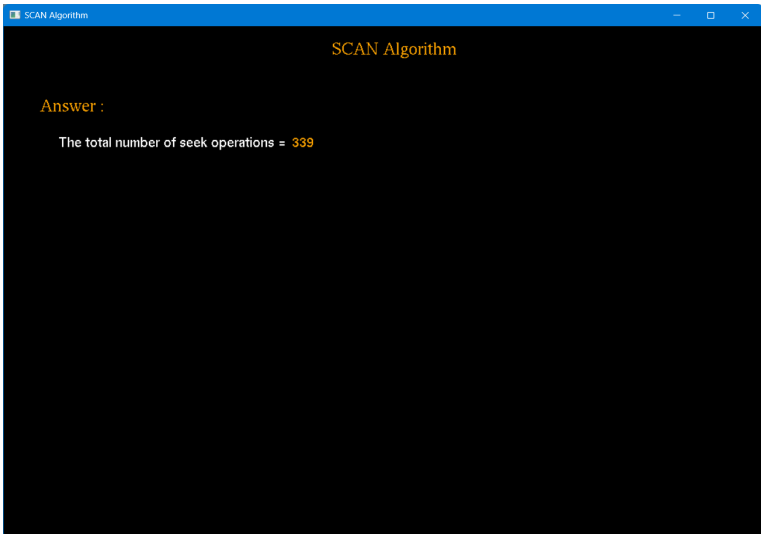

Figure 7.6: Total seek operations - right direction

Chapter 8

Conclusion & Future Enhancements

8.1 Conclusion

This project successfully demonstrates the simulation of the SCAN disk scheduling algorithm using OpenGL. The implementation effectively visualizes the movement of the read-write head across the disk's sectors, providing an intuitive understanding of the SCAN algorithm's operation. By incorporating interactive elements such as mouse and keyboard controls, the project enhances user engagement and learning.

The use of OpenGL for this simulation highlights its capability to create dynamic and visually appealing graphics, making complex algorithms easier to understand. Overall, the project achieves its goal of providing a clear and interactive demonstration of the SCAN disk scheduling algorithm. Future enhancements could include adding support for other disk scheduling algorithms and further refining the user interface to create an even more comprehensive and engaging educational experience.

8.2 Future Enhancements

Future enhancements for this project could include the implementation of additional disk scheduling algorithms, such as FCFS, SSTF, and C-LOOK, to provide a broader comparative analysis. Enhancing the user interface with more interactive features and detailed real-time statistics would further enrich the learning experience. Integrating advanced visualization techniques, such as 3D representations and animations, could also offer a deeper insight into the mechanics of disk scheduling algorithms. Additionally, expanding the project to include a comprehensive tutorial mode would make it an even more valuable educational tool for students and professionals alike.

References

[1] Woo, M., Neider, J., & Davis, T. (2013). OpenGL Programming Guide: The Official Guide to Learning OpenGL (8th ed.). Addison-Wesley.

[2] Moffat, A., & Tanimoto, S. L. (1996). Disk Scheduling Algorithms. ACM Computing Surveys, 28(4), 479-501.

[3] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Operating System Concepts (10th ed.). Wiley.

[4] Hill, D. F., & Kelley, S. M. (2018). Computer Graphics Using OpenGL (4th ed.). Pearson.