VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI-590018

COMPUTER GRAPHICS AND IMAGE PROCESSING (21CSL66)

Mini Project Report

on

Workplace Evacuation Simulation

Submitted in partial fulfillment of the requirements for the VI semester

Computer Science and Engineering of Visvesvaraya Technological University, Belagavi

Submitted by:

Riya	*1RN21CS124*
Shilpa	*1RN21CS144*

Under the Guidance of:

Mrs. S Mamatha Jajur
Assistant 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

2023-2024

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 **Workplace Evacuation Simulation** has been successfully carried out by **Riya** bearing USN **1RN21CS124** and **Shilpa** bearing USN **1RN21CS144**, 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
Mrs. S Mamatha Jajur	**Dr. Kiran P**	**Dr. Ramesh Babu H S**
Assistant 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,**Mrs. S Mamatha Jajur**, Assistant 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

This project is a dynamic simulation of an office layout using OpenGL and C++. It models the movement of individuals (referred to as ”persons”) within a predefined office space, complete with exit doors, entry points, and various room configurations. The simulation supports interactive control over the animation of individuals, allowing them to move towards specified destinations or handle various scenarios like exiting the building or finding entry points. Key features include real-time updating of person positions, handling of person movement using algorithms such as Rapidly-exploring Random Trees (RRT) for pathfinding, and user input handling for controlling animation speed and movement. The system incorporates Cohen-Sutherland line clipping to ensure that movement and visual elements stay within the defined boundaries. This project demonstrates an integration of graphics rendering, real-time simulation, and user interaction to create a comprehensive office environment visualization.

Contents

List of Figures

List of Tables

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:

- **void glutInitDisplayMode(unsigned int mode):**
  This function requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutInitWindowPosition(int x, int y):** This specifies the initial position of top-left corner of the windows in pixels.

- **void glutInitWindowSize(int width, int height):** This function specifies the initial height and width of the window in pixels.

- **void glutCreateWindow(char *title):** This function creates a window on the display the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- **void glutDisplayFunc(void (*func)(void)):** This function registers the display func that is executed when the window needs to be redrawn.

- **void glClearColor(GLclampfr,GLclampfg GLclampfb,GLclampf a):** This sets the present RGBA clear colour used when clearing the colour buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

Chapter 5

Implementation

```
#include <GL/glut.h>

#include <cmath>

#include <iostream>

#include <vector>

#include <limits>

#include<random>


// Constants for the person's movement

#define PERSON_SIZE 10

#define PERSON_COLOR_RED 0.5f

#define PERSON_COLOR_GREEN 0.0f

#define PERSON_COLOR_BLUE 0.5f


// Clipping region boundaries

#define CLIP_LEFT -750

#define CLIP_RIGHT 550

#define CLIP_BOTTOM -550

#define CLIP_TOP 600


// Cohen-Sutherland line clipping region codes

const int INSIDE = 0; // 0000

const int LEFT = 1; // 0001

const int RIGHT = 2; // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8; // 1000
```

```
class Person {
public:
    float x, y;      // Current position
    float destX, destY; // Destination coordinates
    float speed;     // Speed of movement
    bool hasDestination;



    Person(float startX, float startY, float destX, float destY, float speed)
        : x(startX), y(startY), destX(destX), destY(destY), speed(speed) {}
    Person(float startX, float startY, float speed)
        : x(startX), y(startY), speed(speed), hasDestination(false) {}

    // Update the position of the person
    void updatePosition() {
        float dx = destX - x;
        float dy = destY - y;
        float distance = sqrt(dx * dx + dy * dy);

        if (distance > speed) {
            float ratio = speed / distance;
            x += dx * ratio;
            y += dy * ratio;
        }
        else {
            x = destX;
            y = destY;
            hasDestination = false; // Reached the destination
        }
    }

    // Draw the person
    void draw() {
        glColor3f(PERSON_COLOR_RED, PERSON_COLOR_GREEN, PERSON_COLOR_BLUE);
        glPointSize(PERSON_SIZE);
        glBegin(GL_POINTS);
        glVertex2f(x, y);
        glEnd();
    }

    // Set destination coordinates for movement
    void setDestination(float newDestX, float newDestY) {
        destX = newDestX;
        destY = newDestY;
        hasDestination = true;

    }
};

// Global variables
std::vector<Person> persons; // Vector to store all persons
bool animationEnabled = true; // Variable to control animation state

// Exit door coordinates
std::vector<std::pair<float, float>> exitDoors = {
    {-750.0f, 300.0f}, // Exit near Office 1 to Office 2
    {-750.0f, -25.0f}, // Exit near Office 2 to Office 3
    {550.0f, 200.0f}, // Exit near Office 5 to Fax/Copy
    {550.0f, -50.0f} // Exit near Fax/Copy to Office 5
};

// Entry points for each room
std::vector<std::pair<float, float>> entryPoints = {
    {-550.0f, 500.0f}, // Entry point for Office 1
    {-550.0f, 100.0f}, // Entry point for Office 2
    {-550.0f, -25.0f}, // Entry point for Office 3
    {-450.0f, 300.0f}, // Entry point for Conference Room Left
    {-250.0f, 400.0f},   // Entry point for Conference Room Right
    {350.0f, 500.0f}, // Entry point for Lunchroom
    {350.0f, 200.0f}, // Entry point for Fax/Copy
    {350.0f, -50.0f}, // Entry point for Office 5
    {-200.0f, -50.0f}, // Entry point for WC 1
    {-25.0f, -50.0f}, // Entry point for WC 2
    {-375.0f, -50.0f} , // Entry point for Storage
    {-500.0f,120.0f},
    {-450.0f,50.0f},
    {-200.0f,50.0f},
    {350.0f,500.0f}

};

// Function to compute Euclidean distance between two points
float distance(float x1, float y1, float x2, float y2) {
    return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}



void initializePersons() {
    // Office 1 to Office 2
    persons.push_back(Person(-700, 500, -700, 200, 1.0));

    // Office 2 to Office 3
    persons.push_back(Person(-700, 200, -700, -100, 0.5));

    // Conference Room Left to Conference Room Right
    persons.push_back(Person(-400, 200, 0, 400, 0.8));

    // Lunchroom to Fax/Copy
    persons.push_back(Person(400, 500, 400, 200, 0.7));

    // Fax/Copy to Office 5
    persons.push_back(Person(400, 0, 400, -100, 0.4));

    // WC 1 to Storage
    persons.push_back(Person(-200, -100, -250, -250, 0.6));
    // WC 2 to WC 1
    persons.push_back(Person(-50, -100, -50, -250, 0.7));

    // Storage to WC 1
    persons.push_back(Person(-375, -100, -500, -250, 0.5));
}



// Function prototypes
void drawRectangle(float x1, float y1, float x2, float y2);
void drawLine(float x1, float y1, float x2, float y2);
void drawText(float x, float y, const char* string);
void drawRectangleOutline(float x1, float y1, float x2, float y2);
void drawOfficeLayout();
void myinit();
void display();
void animate(int);
void toggleAnimation();
void handleExit();
void keyboard(unsigned char key, int x, int y);



// Function to compute region code for a point (x, y)
int computeCode(float x, float y) {
    int code = INSIDE;

    if (x < CLIP_LEFT)
        code |= LEFT;
    else if (x > CLIP_RIGHT)
        code |= RIGHT;
    if (y < CLIP_BOTTOM)
        code |= BOTTOM;
    else if (y > CLIP_TOP)
        code |= TOP;

    return code;
```

```
}


// Function to perform Cohen-Sutherland clipping
bool cohenSutherlandClip(float& x0, float& y0, float& x1, float& y1) {
    int code0 = computeCode(x0, y0);
    int code1 = computeCode(x1, y1);

    bool accept = false;

    while (true) {
        if ((code0 == 0) && (code1 == 0)) {
            accept = true;
            break;
        }
        else if (code0 & code1) {
            break;
        }
        else {
            int codeOut;
            float x, y;

            if (code0 != 0)
                codeOut = code0;
            else
                codeOut = code1;

            if (codeOut & TOP) {
                x = x0 + (x1 - x0) * (CLIP_TOP - y0) / (y1 - y0);
                y = CLIP_TOP;
            }
            else if (codeOut & BOTTOM) {
                x = x0 + (x1 - x0) * (CLIP_BOTTOM - y0) / (y1 - y0);
                y = CLIP_BOTTOM;
            }
            else if (codeOut & RIGHT) {
                y = y0 + (y1 - y0) * (CLIP_RIGHT - x0) / (x1 - x0);
                x = CLIP_RIGHT;
            }
            else if (codeOut & LEFT) {
                y = y0 + (y1 - y0) * (CLIP_LEFT - x0) / (x1 - x0);
                x = CLIP_LEFT;
            }

            if (codeOut == code0) {
                x0 = x;
                y0 = y;
                code0 = computeCode(x0, y0);
            }
            else {
                x1 = x;
                y1 = y;
                code1 = computeCode(x1, y1);
            }
        }
    }

    return accept;
}


// Function to draw a filled rectangle
void drawRectangle(float x1, float y1, float x2, float y2) {
    glBegin(GL_QUADS);
    glVertex2f(x1, y1);
    glVertex2f(x2, y1);
    glVertex2f(x2, y2);
    glVertex2f(x1, y2);
    glEnd();
}
```

```
// Function to draw a line with clipping
void drawLine(float x1, float y1, float x2, float y2) {
    if (cohenSutherlandClip(x1, y1, x2, y2)) {
        glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glEnd();
    }
}



// Function to draw text at a specific position
void drawText(float x, float y, const char* string) {
    glRasterPos2f(x, y);
    for (const char* c = string; *c != '\0'; c++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, *c);
    }
}



// Function to draw an outline of a rectangle
void drawRectangleOutline(float x1, float y1, float x2, float y2) {
    glBegin(GL_LINES);
    glVertex2f(x1, y1);
    glVertex2f(x2, y1);

    glVertex2f(x2, y1);
    glVertex2f(x2, y2);

    glVertex2f(x2, y2);
    glVertex2f(x1, y2);

    glVertex2f(x1, y2);
    glVertex2f(x1, y1);
    glEnd();
}
```

```
// Function to draw exit doors
void drawExitDoors() {
    glPointSize(8);
    glColor3f(1.0f, 0.0f, 0.0f); // Red color for exit doors

    glBegin(GL_POINTS);
    for (const auto& exitDoor : exitDoors) {
        glVertex2f(exitDoor.first, exitDoor.second);
    }
    glEnd();
}


void drawOfficeLayout() {

    // Set color for walls
    glColor3f(0.1f, 0.5f, 0.7f);

    // Draw external walls
    drawLine(-750, 600, -750, -250);
    drawLine(-750, -250, 550, -250);
    drawLine(550, -250, 550, 600);
    drawLine(-550, 600, 550, 600);

    // Draw office rooms and other spaces with outlines
    drawRectangleOutline(-750, 350, -550, 600);
    drawRectangle(-749, 349, -551, 599);

    drawRectangleOutline(-750, 50, -550, 300);
    drawRectangle(-749, 49, -551, 299);

    drawRectangleOutline(-750, -250, -550, 0);
    drawRectangle(-749, -251, -551, -1);
```

```
drawRectangleOutline(-500, 50, -250, 300);
drawRectangle(-499, 49, -251, 299);

drawRectangleOutline(-250, 50, 300, 600);
drawRectangle(-249, 49, 299, 599);

drawRectangleOutline(350, 350, 550, 600);
drawRectangle(351, 351, 549, 599);

drawRectangleOutline(350, 150, 550, 300);
drawRectangle(351, 149, 549, 299);

drawRectangleOutline(350, -100, 550, 100);
drawRectangle(351, -101, 549, 99);

drawRectangleOutline(-250, -250, -50, -50);
drawRectangle(-249, -251, -51, -49);

drawRectangleOutline(-50, -250, 150, -50);
drawRectangle(-49, -251, 149, -49);

drawRectangleOutline(-500, -250, -250, -50);
drawRectangle(-499, -251, -251, -49);

// Draw table and chairs
glColor3f(0.5f, 0.5f, 0.5f);
drawRectangle(-50, 250, 150, 350);

glColor3f(0.2f, 0.2f, 0.2f);
drawRectangle(-70, 250, -50, 270);
drawRectangle(-70, 330, -50, 350);
drawRectangle(130, 250, 150, 270);
drawRectangle(130, 330, 150, 350);
```

```
glColor3f(0.5f, 0.5f, 0.5f);
drawRectangle(-675, 425, -625, 475);

glColor3f(0.2f, 0.2f, 0.2f);
drawRectangle(-670, 430, -660, 440);
drawRectangle(-670, 470, -660, 460);

glColor3f(0.5f, 0.5f, 0.5f);
drawRectangle(-675, 125, -625, 175);

glColor3f(0.2f, 0.2f, 0.2f);
drawRectangle(-670, 130, -660, 140);
drawRectangle(-670, 170, -660, 160);


glColor3f(0.5f, 0.5f, 0.5f);
drawRectangle(-675, -175, -625, -125);

glColor3f(0.2f, 0.2f, 0.2f);
drawRectangle(-670, -170, -660, -160);
drawRectangle(-670, -130, -660, -140);

glColor3f(0.5f, 0.5f, 0.5f);
drawRectangle(50, 350, 100, 400);

glColor3f(0.2f, 0.2f, 0.2f);
drawRectangle(55, 355, 65, 365);
drawRectangle(95, 355, 85, 365);


glColor3f(0.5f, 0.5f, 0.5f);
drawRectangle(400, 450, 450, 500);

glColor3f(0.2f, 0.2f, 0.2f);

drawRectangle(380, 460, 390, 470);
```

```
    drawRectangle(420, 460, 430, 470);
    drawRectangle(380, 490, 390, 500);


    // Draw entry points as green points
    glColor3f(0.0f, 1.0f, 0.0f);
    glPointSize(7.0f);
    for (const auto& entry : entryPoints) {
        glBegin(GL_POINTS);
        glVertex2f(entry.first, entry.second);
        glEnd();
    }
    //Draw exit door
    drawExitDoors();

    // Labels for rooms
    glColor3f(1.0f, 1.0f, 1.0f);
    drawText(-700, 500, "Office 1");
    drawText(-700, 200, "Office 2");
    drawText(-700, -100, "Office 3");
    drawText(-400, 200, "Conference Room Left");
    drawText(0, 400, "Conference Room Right");
    drawText(400, 500, "Lunchroom");
    drawText(400, 200, "Fax/Copy");
    drawText(400, 0, "Office 5");
    drawText(-200, -100, "WC 1");
    drawText(50, -100, "WC 2");
    drawText(-375, -100, "Storage");
}

// Function to initialize OpenGL settings
void myinit() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-W, W, -H, H);
    glMatrixMode(GL_MODELVIEW);
```

```
}

// Function to display the scene
void display() {
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    drawOfficeLayout();

    // Draw all persons
    for (auto& person : persons) {
        person.draw();
    }



    glFlush();
}

// Function to update person positions and trigger redraw
void animate(int) {
    if (animationEnabled) {
        // Update positions of all persons
        for (auto& person : persons) {
            person.updatePosition();
        }
        glutPostRedisplay();
        glutTimerFunc(1000 / 60, animate, 0);
    }
}



// Function to toggle animation state
void toggleAnimation() {
    animationEnabled = !animationEnabled;
    if (animationEnabled) {
        // Restart animation
        glutTimerFunc(0, animate, 0);
    }
}


// Function to handle exit from nearest exit door
void handleExit() {
    for (auto& person : persons) {
        float minDistance = std::numeric_limits<float>::max();
        std::pair<float, float> nearestExit;

        for (const auto& door : exitDoors) {
            float dx = door.first - person.x;
            float dy = door.second - person.y;
            float distance = sqrt(dx * dx + dy * dy);

            if (distance < minDistance) {
                minDistance = distance;
                nearestExit = door;
            }
        }

        // Set person's destination to the nearest exit
        person.setDestination(nearestExit.first, nearestExit.second);
    }
}


void handle() {
    srand(time(nullptr)); // Seed random number generator

    static std::vector<int> currentEntryPointIndex(persons.size(), 0); // Track the
        current entry point index for each person

    for (size_t i = 0; i < persons.size(); ++i) {
        auto& person = persons[i];
        int& currentIndex = currentEntryPointIndex[i];
        // Check distances to all entry points
        float minDistance = std::numeric_limits<float>::max();
        std::pair<float, float> nearestEntry;

        for (size_t j = 0; j < entryPoints.size(); ++j) {
            float dist = distance(person.x, person.y, entryPoints[j].first,
                entryPoints[j].second);

            if (dist < minDistance) {
                minDistance = dist;
                nearestEntry = entryPoints[j];
                currentIndex = j; // Update the current index to the nearest entry
                    point
            }
        }

        // Set person's destination to the nearest entry point
        person.setDestination(nearestEntry.first, nearestEntry.second);
    }
}



void handleEntry() {
    const int maxIterations = 1000;
    const float stepSize = 20.0f;
    const float goalThreshold = 30.0f;

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<float> disX(CLIP_LEFT, CLIP_RIGHT);
    std::uniform_real_distribution<float> disY(CLIP_BOTTOM, CLIP_TOP);

    for (auto& person : persons) {
        // Skip if the person already has a destination
```

```
if (person.hasDestination) continue;

// Initialize RRT structure for the current person
std::vector<std::pair<float, float>> tree;
tree.push_back(std::make_pair(person.x, person.y)); // Start with the
    person's current position

// Expand the tree
for (int i = 0; i < maxIterations; ++i) {
    // Generate a random point
    float randomX = disX(gen);
    float randomY = disY(gen);
    std::pair<float, float> randomPoint(randomX, randomY);

    // Find the closest point in the tree
    float minDist = std::numeric_limits<float>::max();
    std::pair<float, float> closestPoint;
    for (const auto& node : tree) {
        float dist = distance(node.first, node.second, randomPoint.first,
            randomPoint.second);
        if (dist < minDist) {
            minDist = dist;
            closestPoint = node;
        }
    }

    // Move from the closest point towards the random point
    float dx = randomPoint.first - closestPoint.first;
    float dy = randomPoint.second - closestPoint.second;
    float distToRandom = sqrt(dx * dx + dy * dy);

    if (distToRandom > stepSize) {
        float newX = closestPoint.first + (dx / distToRandom) * stepSize;
        float newY = closestPoint.second + (dy / distToRandom) * stepSize;
        tree.push_back(std::make_pair(newX, newY));
    }
```

```
            else {
                tree.push_back(randomPoint);
            }

            // Check if we have reached any entry point
            for (size_t j = 0; j < entryPoints.size(); ++j) {
                float dist = distance(tree.back().first, tree.back().second,
                    entryPoints[j].first, entryPoints[j].second);
                if (dist < goalThreshold) {
                    // Update the person's destination to this entry point
                    person.setDestination(entryPoints[j].first,
                        entryPoints[j].second);
                    break; // Exit loop once the goal is reached
                }
            }

            if (person.hasDestination) {
                break; // Exit loop if the person has a destination
            }
        }
    }
}


void keyboard(unsigned char key, int x, int y) {
    switch (key) {
    case ' ':
        toggleAnimation();//pause and move
        break;
    case 's':handleEntry(); //Move towards Nearer Entry Points
        break;
    case 'S':handle();//Move towards Random Entry Points
        break;
    case 'd':
        handleExit();//Move towards Exit Points
        break;
```

```
case '1':
    if (!persons.empty()) {
        persons[0].setDestination(100.0f, 100.0f);
    }
    break;
case '2':
    if (persons.size() > 1) {
        persons[1].setDestination(-100.0f, -100.0f); //person 1 moves from one
            room to another
    }
    break;
case '3':
    if (persons.size() > 2) {
        persons[2].setDestination(200.0f, 200.0f); //person 2 moves from one
            room to another
    }
    break;
case '+':
    for (auto& person : persons) {
        person.speed += 0.1f;//speeed of person increases
    }
    break;
case '-':
    for (auto& person : persons) {
        if (person.speed > 0.1f) {
            person.speed -= 0.1f;//speeed of person decreases
        }
    }
    break;
case 'r':
    initializePersons();//adds more people
    break;
case 'c':
    persons.clear();//clears entire office
    break;
case 'a':
```

```
            persons.push_back(Person(0.0f, 0.0f, 100.0f, 100.0f, 0.5f)); //adds one
                person
            break;
        default:exit(0);
            break;
        }
}



// Main function
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(W, H);
    glutCreateWindow("Office Layout with Routes");
    myinit();
    initializePersons();
    glutDisplayFunc(display);
    glutTimerFunc(0, animate, 0);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

Chapter 6

Testing

Unit testing is crucial for ensuring that individual components of the system work as expected. This chapter details the unit testing process and results for the OpenGL office layout simulation program. The tests focus on various functionalities of the program, including person movement, rendering, and keyboard controls. The following table summarizes the test cases and their outcomes.

6.1 Testing Procedure

To execute the test cases, follow these steps:

1. Compile the program using the provided commands.
2. Run the program and interact with the simulation using the specified keyboard controls.
3. Observe the behavior of persons, movement, and other functionalities as described in the test cases.
4. Document any deviations from the expected results and adjust the code accordingly.

6.2 Test Case Validation

The following table provides details on the validation of different functionalities within the simulation program.

Table 6.1: Test Case Validation

Test Case No.	Metric	Description	Observation
1	Person Initialization	Checks if persons are initialized correctly with given coordinates and speeds.	Results are as expected. Persons are placed correctly with specified properties.
2	Movement	Tests if persons move according to their speed and destination.	Persons move towards their destinations with smooth animation and at expected speeds.
3	Keyboard Controls	Validates the functionality of keyboard inputs for controlling simulation (e.g., toggle animation, set destinations).	Keyboard controls work as expected, changing person destinations and toggling animation correctly.
4	Clipping	Tests the Cohen-Sutherland clipping algorithm for correctly handling the viewport clipping of lines and shapes.	Clipping works correctly, and shapes are rendered within the visible area of the viewport.
5	Speed Adjustment	Checks if adjusting speed affects the movement speed of all persons.	Speed adjustment works correctly, and persons' movement speeds are updated as expected.
6	Initialization	Tests the reinitialization and clearing of persons in the simulation.	Persons are correctly reinitialized or cleared from the scene based on user input.

Chapter 7

7.1 Results & Snapshots

Figure 7.1 shows the main screen of Office Environment.

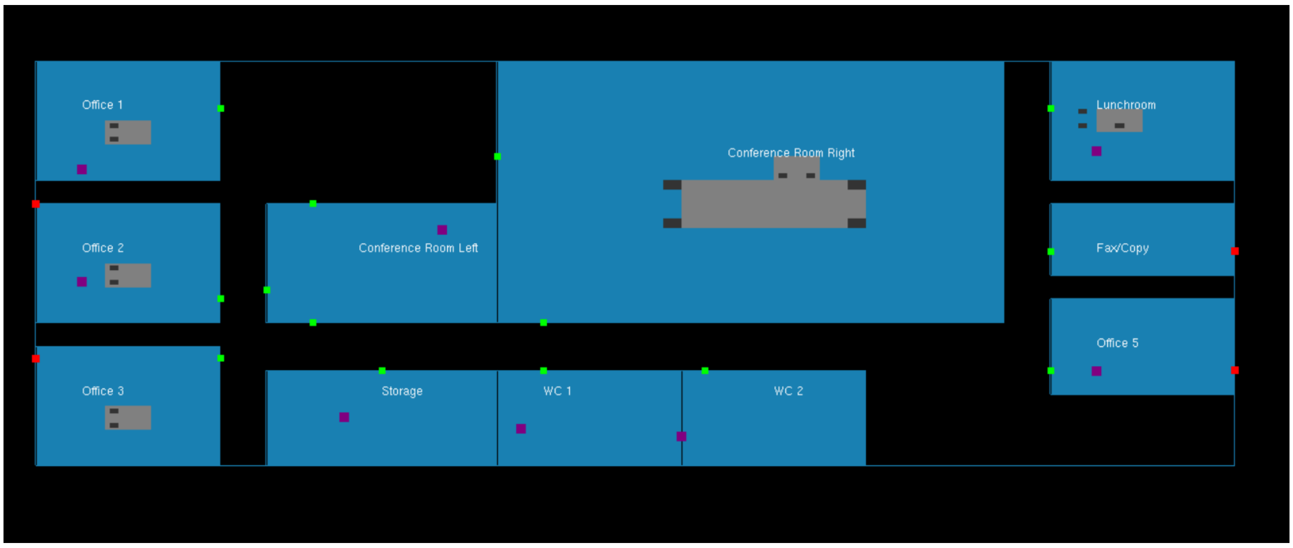

Figure 7.1: Office Environment where people are moving

Figure 7.2 shows People moving towards their Nearer Entry Points when ‘S’ button is clicked in the keyboard

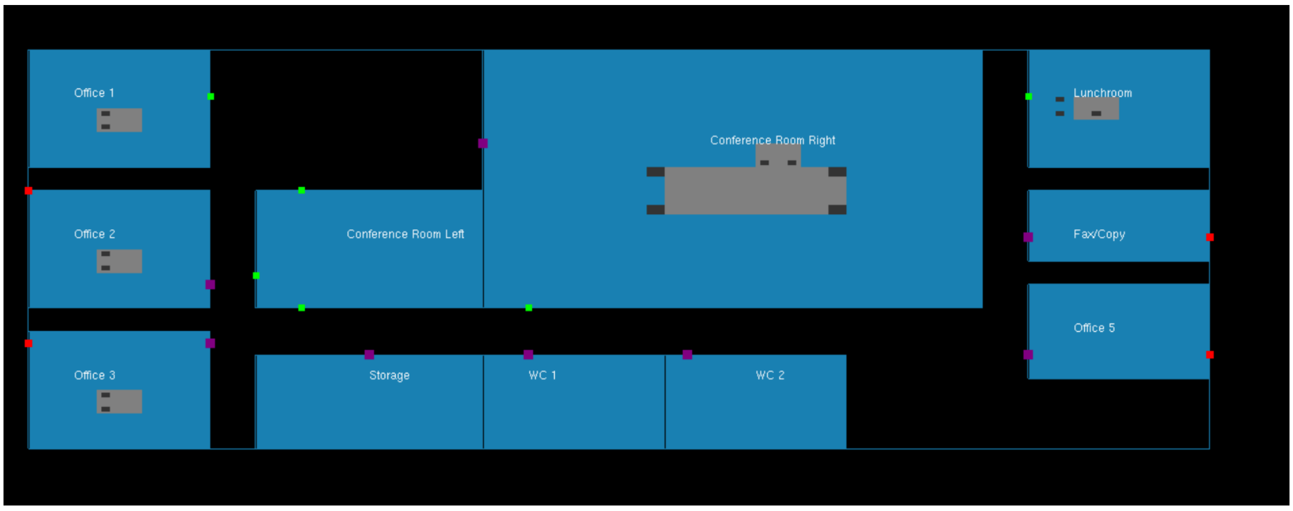

Figure 7.2: People at the Entry Point

Figure 7.3 shows People moving towards Random Entry Points when ‘s’ button is clicked in the keyboard

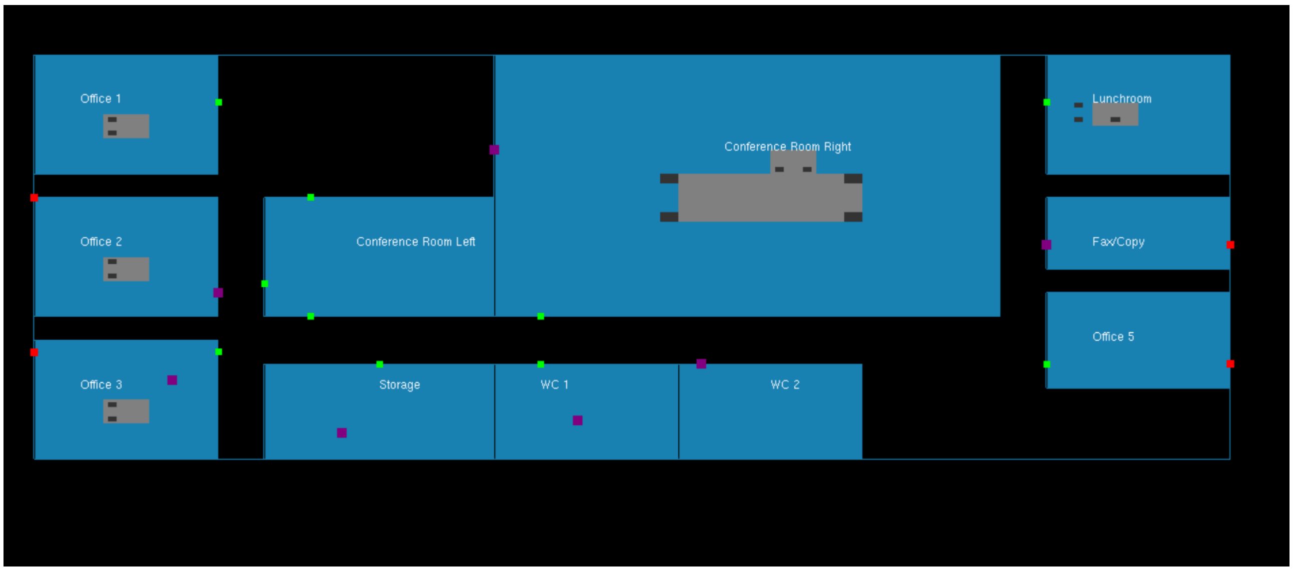

Figure 7.3: People moving towards Random Entry Point

Figure 7.4 shows People moving towards Nearer Exit Points (during danger situation) when 'd' button is clicked in the keyboard

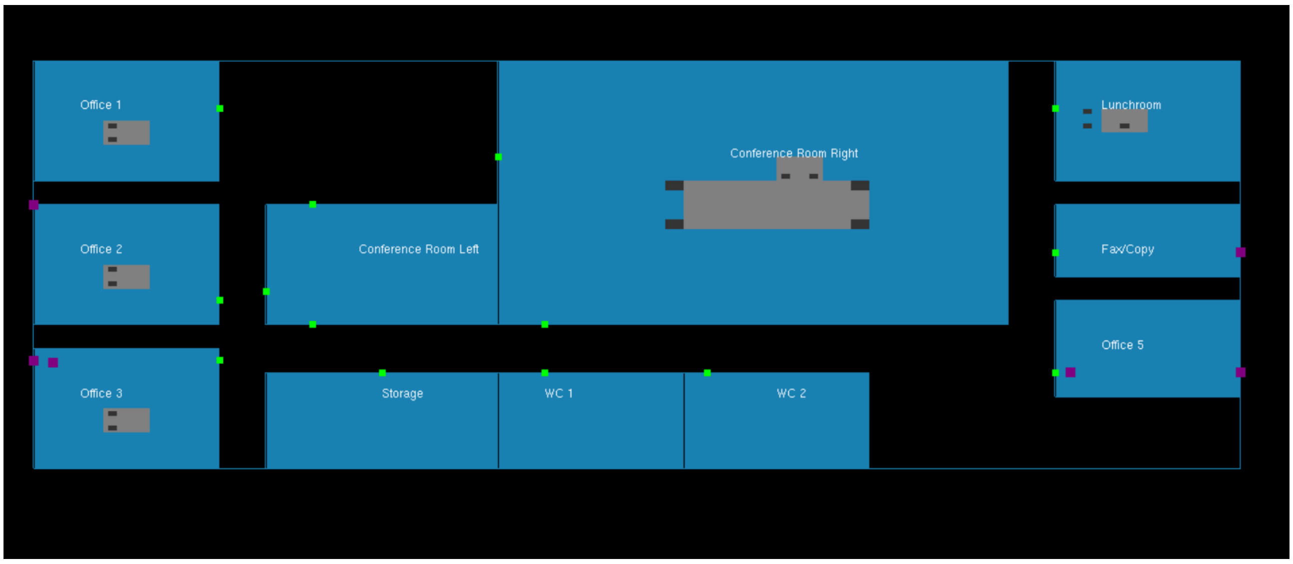

Figure 7.4: People moving towards their Nearer Exit Points

Figure 7.5 shows that Number of People increases when 'r' button is clicked in the keyboard

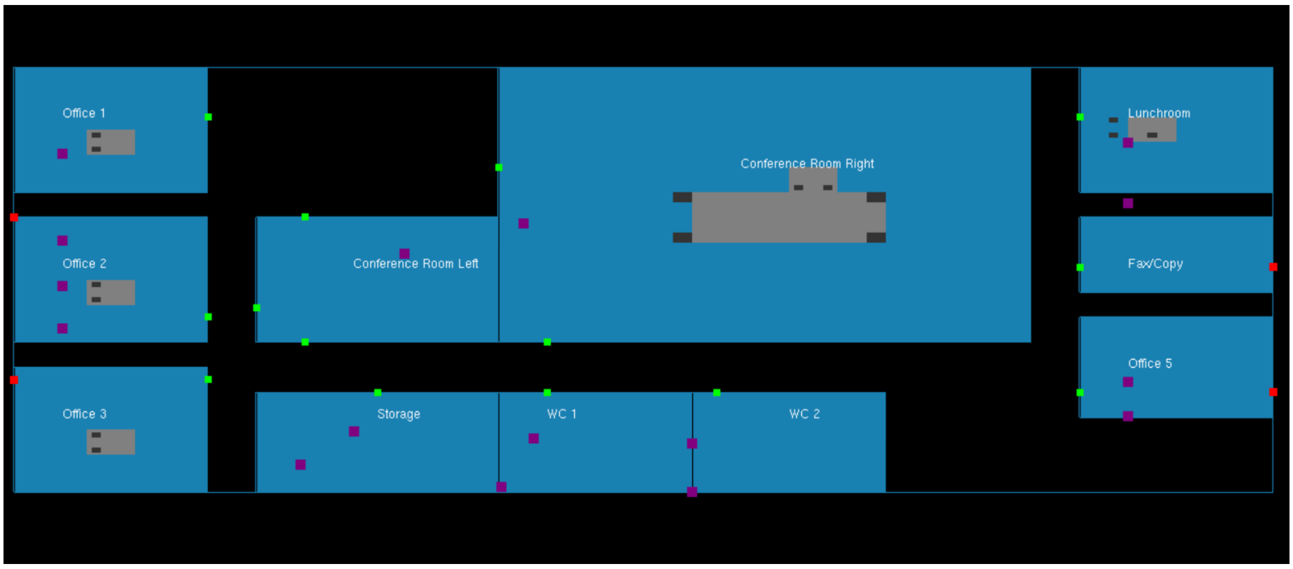

Figure 7.5: More People moving

Figure 7.6 shows that Office will become Empty when 'c' button is clicked in the keyboard

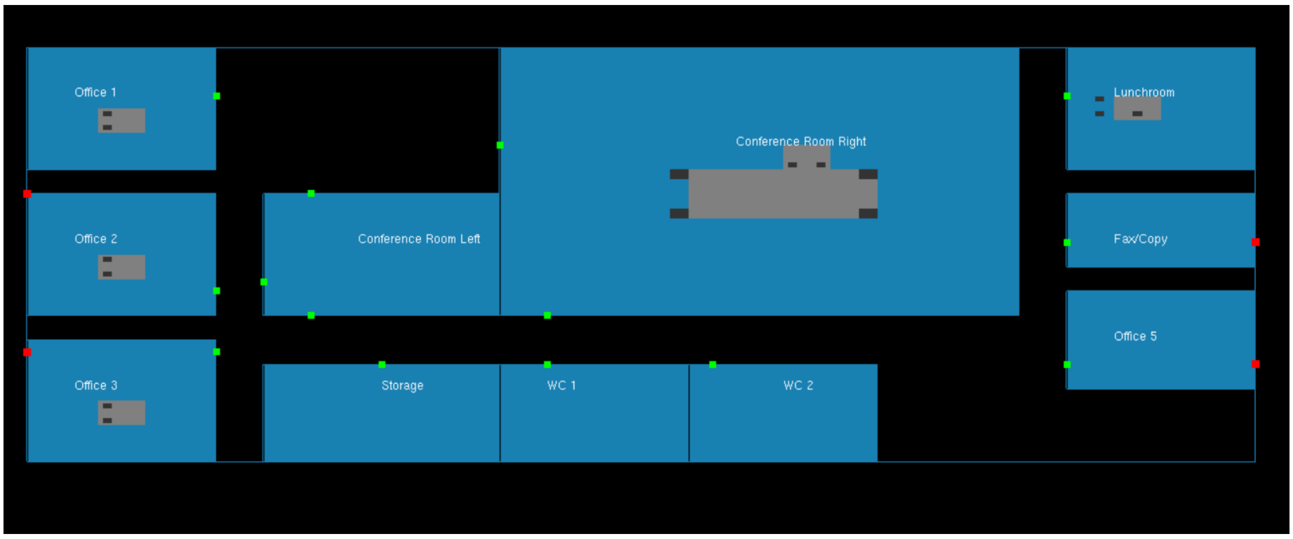

Figure 7.6: Empty Office Environment

Chapter 8

Conclusion & Future Enhancements

8.1 Conclusion

The simulation of a workplace environment using OpenGL and C++ has successfully demonstrated the dynamic movement and behavior of individuals within a predefined office layout. By integrating various functionalities such as realistic movement, collision detection, and optimized evacuation routes, this project provides a comprehensive visualization of how people navigate through an office space. The implementation of the Cohen-Sutherland line clipping algorithm and the use of random point generation for the Rapidly-Exploring Random Tree (RRT) method have enhanced the simulation's accuracy and efficiency. Overall, this project highlights the potential of computer graphics in modeling and analyzing real-world scenarios, offering valuable insights into workspace optimization and safety measures.

8.2 Future Enhancements

The project can include implementing advanced pathfinding algorithms like A* or D* Lite for optimized evacuation routes, incorporating real-time interaction for dynamic adjustments, and adding complex behavioral models for realistic movement simulation. Transitioning to 3D visualization will offer a more immersive experience, while integrating real-world data will enhance practical applicability. Improving scalability for larger environments and adding Virtual Reality (VR) support will further extend the simulation's usability, making it a powerful tool for workspace analysis, safety planning, and virtual training.

References

[1] Edward Angel, *“Interactive Computer Graphics A Top-Down Approach With OpenGL”* 5th Edition, Addison-Wesley, 2008.

[2] F.S. Hill,*“Computer Graphics Using OpenGL”*, 2nd Edition, Pearson Education, 2001.

[3] James D.Foley, Andries Van Dam, Steven K. Feiner, John F Hughes, *”Computer Graphics”*, Second Edition, Addison-Wesley Professional, August 14,1995.

[4] @online OpenGL Official ,`https://www.opengl.org/`

[5] @online OpenGL Overview , `https://https://www.khronos.org/opengl/`