Table of Contents

List of Figures

ABSTRACT

In today's digital age, security in client-server communication is paramount. Encryption and decryption play a crucial role in safeguarding sensitive data exchanged between clients and servers. This project presents a simulation of client-server architecture implemented in OpenGL using C, focusing on the encryption and decryption processes. The client-server model facilitates efficient communication between distributed systems, with the server acting as a central repository of resources and the client accessing these resources remotely. The OpenGL framework provides a powerful platform for creating interactive graphical applications, enabling visualization of the encryption and decryption algorithms in action. This project aims to demonstrate the principles of encryption and decryption through a visually engaging simulation. The client application initiates secure communication with the server, transmitting encrypted data using established cryptographic techniques. Upon receiving the encrypted data, the server performs decryption using corresponding keys, ensuring the confidentiality and integrity of the transmitted information.

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the completion of our project.

We would like to thank **Shri. Narayan Rao R Maanay**, Secretary, BNMEI, Bengaluru for providing the excellent environment and infrastructure in the college.

We would like to sincerely thank **Prof. T J Rama Murthy**, Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

We would like to sincerely thank **Dr. S Y Kulkarni** ,Additional Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

We would like to express my gratitude to **Prof. Eishwar N Maanay**, Dean, BNMIT, Bengaluru for his relentless support and encouragement.

We would like to thank **Dr. Krishnamurthy G N**, Principal, BNMIT, Bengaluru for his constant encouragement.

We would like to thank, **Dr. Chayadevi M L**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

We would also like to thank **Prof. Akshitha Katkeri,** Assistant Professor, Department of Computer Science and Engineering for providing me with her valuable insight and guidance wherever required throughout the course of the project and its successful completion.

Skanda J
1BG21CS127
Srujan Raghavendra S
1BG21CS130

Chapter 1

INTRODUCTION

1.1 Overview

In an era defined by digital connectivity, client-server architectures form the backbone of communication in distributed systems. From online banking to social media interactions, these architectures enable seamless exchanges of data between users and remote servers. However, with this convenience comes the ever-present challenge of safeguarding sensitive information from potential threats. Encryption and decryption emerge as pivotal tools in the arsenal of cybersecurity, offering a shield against unauthorized access and data breaches. Through complex algorithms, encryption scrambles data into an unreadable format, while decryption reverses this process, allowing authorized parties to access the original information securely. Our project embarks on a journey to explore the intricacies of client-server architecture, with a special focus on the encryption and decryption processes.

1.2 Problem Statement

In contemporary distributed systems, ensuring secure data exchange between clients and servers is paramount. However, there is a lack of accessible tools that offer a visual understanding of encryption and decryption processes within a client-server framework. Existing resources often fail to provide an interactive and engaging learning experience, hindering users' comprehension of these crucial security concepts.

1.3 Motivation

In an era marked by increasing digital interconnectedness, the security of data exchanged between clients and servers is paramount. Whether it's personal communications, financial transactions, or sensitive business information, the integrity and confidentiality of this data are constantly at risk from cyber threats. By delving into the intricacies of encryption and decryption within client-server architecture, individuals and organizations can fortify their defences against these threats. The motivation behind this project is to empower individuals and organizations to navigate the digital landscape with confidence and resilience.

1.4 Computer Graphics

Computer graphics and multimedia technologies are becoming widely used in educational applications because they facilitate non-linear, self-learning environments that particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as CGI. Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modelling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others.

The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.5 OpenGL API

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released it on June 30, 1992; applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games. Since 2006 OpenGL has been managed by the non-profit technology consortium Khronos Group.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware. The API is defined as a set of functions which

may be called by the client program, alongside a set of named integer constants. In addition to being language-independent, OpenGL is also cross-platform.

Given that creating an OpenGL context is quite a complex process, and given that it varies between operating systems, automatic OpenGL context creation has become a common feature of several game-development and user-interface libraries, including SDL, Allegro, SFML, FLTK, and Qt. A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was OpenGL Utility Toolkit (GLUT), later superseded by freeglut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

Per Vertex Operations:

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

Primitive Assembly:

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

Pixel Operation:

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Nextthe data is scaled, biased, and processed by

a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Colour and depth values are assigned for each fragment square.

Fragment Operations:

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

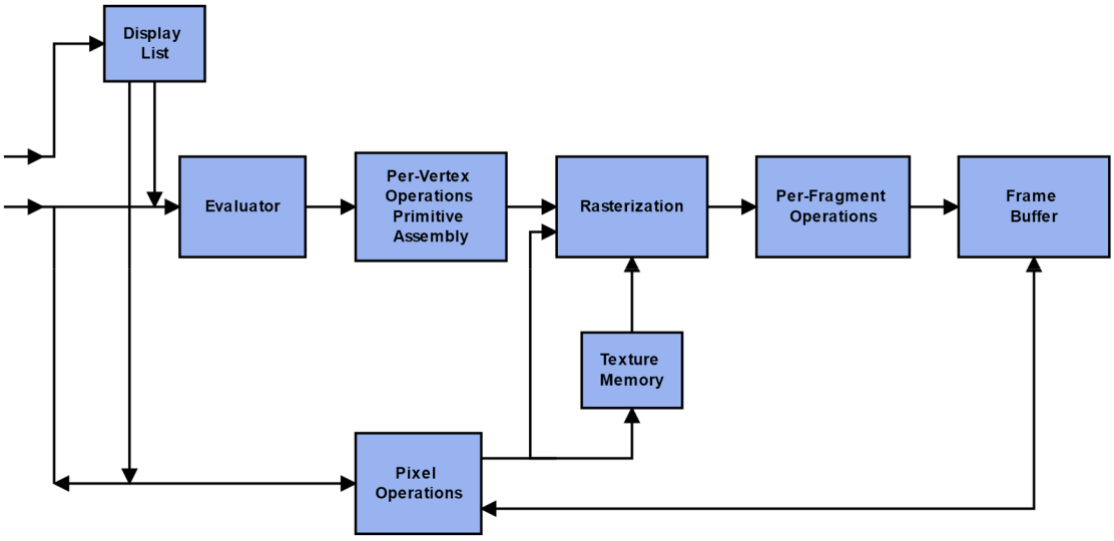

Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain.

We can classify applications of computer graphics into four main areas:

1.6.1 Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering.

students have become familiar with interpreting data plotted on log paper. More recently, software packages that allow interactive design of charts incorporating colour, multiple data sets, and alternate plotting methods have become the norm. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

1.6.2 Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

1.6.3 Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real time. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, such as a mouse, that allows him to communicate with the workstation.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The term "computer graphics" was coined in 1960 by William Fetter, a designer at Boeing, to describe his own job, the field can be said to have first arrived with the publication in 1963 of Ivan Sutherland's Sketchpad program, as part of his Ph.D. thesis at MIT. Sketchpad, as its name suggests, was a drawing program. Beyond the interactive drawing of primitives such as lines and circles and their manipulation – in particular, copying, moving and constraining – with use of the then recently invented light pen, Sketchpad had the first fully-functional graphical user interface (GUI) and the first algorithms for geometric operations such as clip and zoom. Interesting, as well, is that Sketchpad's innovation of an object-instance model to store data for geometric primitives foretold object-oriented programming. Coincidentally, on the hardware side, the year 1963 saw the invention by Douglas Engelbart at the Stanford Research Institute of the mouse, the humble device even today carrying so much of GUI on its thin shoulders.

Subsequent advances through the sixties came thick and fast: raster algorithms, the implementation of parametric surfaces, hidden-surface algorithms and the representation of points by homogeneous coordinates, the latter crucially presaging the foundational role of projective geometry in 3D graphics, to name a few. Flight simulators were the killer app of the day and companies such as General Electric and Evans & Sutherland, 6 co-founded by Douglas Evans and Ivan Sutherland, wrote simulators with real-time graphics.

The seventies, brought the Z-buffer for hidden surface removal, texture mapping, Phong's lighting model – all crucial components of the OpenGL API (Application Programming Interface) we'll be using soon – as well as keyframe-based animation. Photorealistic rendering of animated movie keyframes almost invariably deploys ray tracers, which were born in the seventies too.Through the nineties, as well, the use of 3D effects in movies became pervasive. The Terminator and Star Wars series, and Jurassic Park, were among the early movies to set the standard for CGI. Toy Story from Pixar, 8 released in 1995, has special importance in the history of 3D CGI as the first movie to be entirely computer-generated – no scene was ever pondered through a glass lens, nor any recorded on a photographic reel! It was cinema without film.

Quake, released in 1996, the first of the hugely popular Quake series of games, was the first fully 3D game.

Another landmark from the nineties of particular relevance to us was the release in 1992 of OpenGL, the open-standard cross-platform and cross language 3D graphics API, by Silicon Graphics. OpenGL is actually a library of calls to perform 3D tasks, which can be accessed from programs written in various languages and running over various operating systems. That OpenGL was high-level (in that it frees the applications programmer from having to care about such low-level tasks as representing primitives like lines and triangles in the raster, or rendering them to the window) and easy to use (much more so than its predecessor 3D graphics API, PHIGS, standing for Programmer's Hierarchical Interactive Graphics System) first brought 3D graphics programming to the "masses". What till then had been the realm of a specialist was now open to a casual programmer following a fairly amicable learning curve.

Since its release OpenGL has been rapidly adopted throughout academia and industry. It's only among game developers that Microsoft's proprietary 3D API, Direct3D, which came soon after OpenGL bearing an odd similarity to it but optimized for Windows, is more popular.

The story of the past decade has been one of steady progress, rather than spectacular innovations in CG. Hardware continues to get faster, better, smaller and cheaper, continually pushing erstwhile high-end software down market, and raising the bar for new products. The almost complete displacement of CRT monitors by LCD and the emergence of high-definition television are familiar consequences of recent hardware evolution.

2.2 Related Work

- **Computer Aided Design (CAD):**

  Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kinds of Industries. Its use in designing electronic systems is known as electronic design automation (EDA). In mechanical design it is known as mechanical design automation (MDA) or computer-aided drafting (CAD), which includes the process of creating a technical drawing with the use of computer software.

- **Computer Simulation:** Computer simulation is the reproduction of the behaviour of a system using a computer to simulate the outcomes of a mathematical model associated with said system. Since they allow to check the reliability of chosen mathematical models,

computer simulations have become a useful tool for the mathematical modelling of many natural systems in physics (computational physics), astrophysics, climatology, chemistry, biology and manufacturing, human systems in economics, psychology, social science, health care and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

- **Digital Art**: Digital art is an artistic work or practice that uses digital technology as part of the creative or presentation process. Since the 1970s, various names have been used to describe the process, including computer art and multimedia art. Digital art is itself placed under the larger umbrella term new media art. With the rise of social media and the internet, digital art application of computer graphics. After some initial resistance, the impact of digital technology has transformed activities such as painting, drawing, sculpture and music/sound art, while new forms, such as net art, digital installation art, and virtual reality, have become recognized artistic practices. More generally the term digital artist is used to describe an artist who makes use of digital technologies in the production of art. In an expanded sense, "digital art" is contemporary art that uses the methods of mass production or digital media.

- **Virtual Reality**: Virtual reality (VR) is an experience taking place within a computer-generated reality of immersive environments can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. gaming) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality. Currently standard virtualreality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual environment. A person using virtual reality equipment is able to look around the artificial world, move around in it, and interact with virtual features or items. The effect is commonly created by VR headsets consisting of a head-mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens. Virtual reality typically incorporates auditory and video feedback, but may also allow other types of sensory and force feedback through haptic technology.

- **Video Games**: A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a two- or three-dimensional video display device such as a TV screen, virtual reality headset or computer monitor. Since the 1980s, video games have become an increasingly important part of the entertainment industry, and whether they are also a form of art is a matter of dispute. The electronic systems used to play video games are called platforms. Video games are developed and released for one or several platforms and may not be available on others. Specialized platforms such as arcade games, which present the game in a large, typically coin-operated chassis, were common in the 1980s in video arcades, but declined in popularity as other, more affordable platforms became available. These include dedicated devices such as video game consoles, as well as general-purpose computers like a laptop, desktop or handheld computing devices.

Chapter 3

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

The following are the software requirements for the application:

- Operating System: Windows 10/11
- Compiler: GNU C/C++ Compile
- Development Environment: Visual Studio 2019 Community Edition
- API: OpenGL API & Win32 API for User Interface and Interaction

3.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware.

- CPU: Intel or AMD processor
- Cores: Dual-Core (Quad-Core recommended)
- RAM: minimum 4GB (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

- Client-Server Architecture: The system will adhere to the client-server model where the client initiates requests and the server responds to these requests. This architecture enables efficient communication and allows for the implementation of security measures at both ends.
- Encryption and Decryption Modules: The core components of the system will include encryption and decryption modules implemented using appropriate cryptographic algorithms. These modules will handle the encryption of data before transmission from the client to the server and the decryption of received data on the server side.
- User Interface: The system will feature a graphical user interface (GUI) implemented using OpenGL. The GUI will provide visual representations of the encryption and decryption processes, allowing users to interact with encryption parameters and observe the effects in real-time.
- Networking: Networking functionality will be implemented to facilitate communication between the client and server over a network. This will involve establishing connections, sending encrypted data from the client to the server, and receiving decrypted data on the server side.
- Encryption Key Management: The system will include mechanisms for managing encryption keys securely. This may involve generating and distributing encryption keys, as well as implementing key exchange protocols to ensure confidentiality and integrity.
- Error Handling and Logging: Error handling mechanisms will be incorporated to detect and handle any exceptions or failures that may occur during encryption, decryption, or network communication. Additionally, logging functionality will be implemented to record system events and errors for troubleshooting and audit purposes.
- Documentation and Tutorials: The system will be accompanied by comprehensive documentation and tutorials to guide users through the setup, configuration, and usage of the system. This will include explanations of encryption concepts, step-by-step instructions for using the GUI, and troubleshooting tips.
- Security Measures: The system will prioritize security by implementing best practices for secure communication, such as using secure protocols (e.g., TLS/SSL), ensuring data

integrity through message authentication codes (MACs), and protecting against attacks like replay attacks and man-in-the-middle attacks.

4.2 Data Flow Diagram

- Accept Mouse Input: The system begins by accepting mouse input from the user. This input is used to determine the action the user wants to perform, whether it's selecting the client-server architecture window or the encryption-decryption window.
- Display Corresponding Window: Based on the mouse input received, the system displays the corresponding window. If the user selects the client-server architecture window, the interface for configuring client-server communication parameters is displayed. If the user selects the encryption-decryption window, the interface for performing encryption and decryption operations is shown.
- Client-Server Architecture Window: Accept Mouse Input for Protocol Selection: Within the client-server architecture window, the system accepts mouse input to allow the user to select the type of protocol for data transmission. This could include options such as TCP/IP, UDP, HTTP, etc.
- Perform Actions Based on Protocol Selection: Depending on the protocol selected by the user, the system performs the necessary actions to configure the client-server communication accordingly. This may involve setting up sockets, defining communication ports, and establishing connections.
- Encryption-Decryption Window: Accept Keyboard Input: In the encryption-decryption window, the system accepts keyboard input from the user. This input typically consists of the plaintext data that the user wants to encrypt or ciphertext that the user wants to decrypt.
- Perform Encryption or Decryption Process: Upon receiving the keyboard input, the system executes the corresponding encryption or decryption process based on the user's action. This involves applying cryptographic algorithms (e.g., AES, RSA) and encryption keys to encrypt or decrypt the data.
- Display Encrypted or Decrypted Result: After performing the encryption or decryption process, the system displays the resulting ciphertext or plaintext to the user, respectively.

4.2 Flowchart

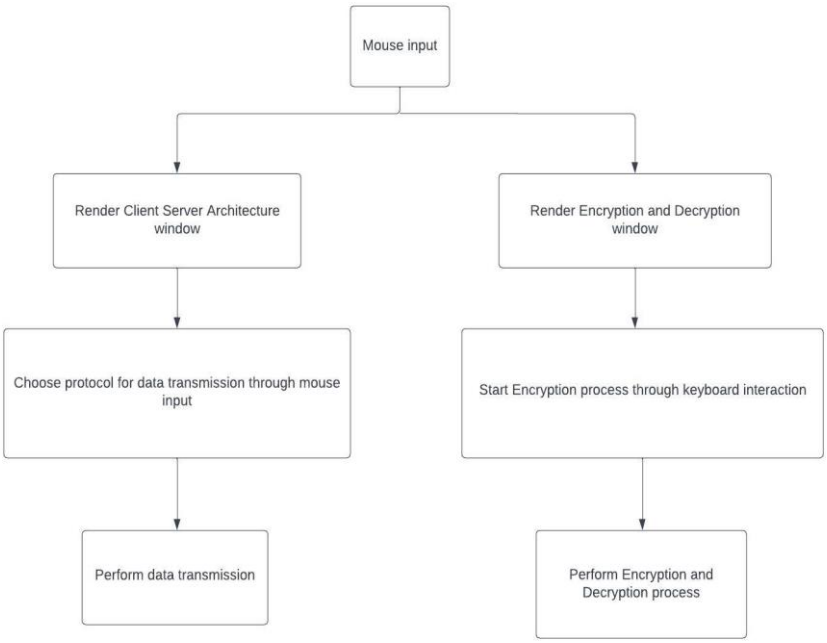

Figure 4.2 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **Graphical Representation:** The simulation provides a graphical representation of the network environment, including the clients, server, and communication pathways.
- **Client-Server Interaction:** Users can observe the interaction between clients and the server, including the handshake process, data upload, and data download.
- **Dynamic Animation:** The movement of elements such as clients, server, and data transfer processes are animated dynamically, providing a realistic visualization of network communication.
- **draw(float radius):** Draws a circle with the specified radius using OpenGL commands.
- **Init():** Initializes the display settings, clears buffers, and sets up projection matrices for 2D rendering.
- **display():** Main display function that calls other functions to render the entire scene.
- **reshape(int w, int h):** Called when the window is resized to adjust the viewport accordingly.
- **main(int argc, char** argv):** Entry point of the program that initializes GLUT, creates the main window, sets up callbacks, and enters the GLUT main loop.
- **myinit():** Initializes OpenGL settings such as background color and shading model.
- **drawScene():** Renders the entire scene, including the circle and the text.
- **setOrthographicProjection():** Sets up an orthographic projection matrix for 2D rendering.
- **resetPerspectiveProjection():** Resets the projection matrix to a perspective projection.
- **renderBitmapString(float x, float y, void* font, const char* string):** Renders a string of text at the specified coordinates using bitmap fonts.

5.2 High Level Code

- **Include Libraries:** The code includes necessary libraries such as OpenGL, GLUT (OpenGL Utility Toolkit), and standard input/output (stdio.h).

- **Constants:**
  - radius: Specifies the radius of the circle to be drawn.
  - text[]: Stores the text string to be displayed on the screen.

- **initGL() Function:**
- This function initializes various OpenGL settings.
- It sets the background color to black using glClearColor().
- It sets the shading model to smooth using glShadeModel().

- **drawCircle() Function:**
- This function draws a circle using OpenGL primitives.
- It takes two parameters: r for the radius of the circle and num_segments for the number of line segments used to approximate the circle.
- It iterates through each segment and calculates the vertex positions using trigonometric functions (sin() and cos()) to create a circular shape.

- **drawText() Function:**
- This function renders text on the screen using OpenGL's bitmap font rendering functionality.
- It takes two parameters: x and y for the position of the text.
- It iterates through each character in the text[] string and renders it on the screen using glutBitmapCharacter().

- **display() Function:**
- This function is the display callback function for GLUT.
- It clears the color buffer using glClear() with the GL_COLOR_BUFFER_BIT flag.
- It sets up the projection matrix using glMatrixMode() and glLoadIdentity().

- It calls drawCircle() to draw the circle and drawText() to render the text on the screen.
- Finally, it swaps the buffers using glutSwapBuffers().

- **reshape() Function:**
- This function is the reshape callback function for GLUT.
- It adjusts the viewport to match the new window size using glViewport().
- It sets up the projection matrix using glMatrixMode() and glLoadIdentity().

- **main() Function:**
- This is the main entry point of the program.
- It initializes GLUT using glutInit() and sets the display mode using glutInitDisplayMode().
- It creates the window using glutCreateWindow() and sets the window title.
- It registers the display and reshape callback functions using glutDisplayFunc() and glutReshapeFunc().
- Finally, it enters the GLUT event processing loop using glutMainLoop().

5.2.1 User Implementation

- **Reshape Function**

```
void reshape(int w, int h)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 400.0, 0.0, 400.0);
}
```

- **Display Function**

```
void display()
{
        glPushMatrix();
        line();
        glPopMatrix();

        glPushMatrix();
        glTranslatef(0.0, 220.0, 0.0);
        line();
        glPopMatrix();
        glPushMatrix();
        server();
        glPopMatrix();

        glPushMatrix();
        glColor3f(0.22, 0.22, 0.22);
        glTranslatef(200.0, 140.0, 0.0);
        draw(7.0);
        glPopMatrix();

        glPushMatrix();
        glColor3f(0.0, 0.0, 0.0);
        glTranslatef(200.0, 140.0, 0.0);
        draw(5.0);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(100.0, 0.0, 0.0);

char* ptr5 = "Client 2";
        int len5 = strlen(ptr5);
        glColor3f(0.3, 0.3, 0.3);
        glRasterPos3f(-53, 20, 0.0);
```

```
for (int i = 0; i < len5; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, ptr5[i]);
glPopMatrix();
glPushMatrix();
glTranslatef(100.0, 0.0, 0.0);
char* ptr1 = "Client 1";
int len1 = strlen(ptr1);
glColor3f(0.3, 0.3, 0.3);
glRasterPos3f(-53, 240, 0.0);
for (int i = 0; i < len1; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, ptr1[i]);
glPopMatrix();
glPushMatrix();
glTranslatef(100.0, 0.0, 0.0);
char* ptr7 = "SERVER";
int len7 = strlen(ptr1);
glColor3f(0.3, 0.3, 0.3);
glRasterPos3f(90, 85, 0.0);
for (int i = 0; i < len7; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, ptr7[i]);
glPopMatrix();
glPushMatrix();
one();
glPopMatrix();
glPushMatrix();
two();
glPopMatrix();
glPushMatrix();
if (count == 1)
{
        three();
}
glPopMatrix();
glPushMatrix();
```

```
        if (count1 == 1)
        {
                four();
        }
        glPopMatrix();

        glPushMatrix();

if (dt == 1)
        {
                five();
        }
        glPopMatrix();

        glPushMatrix();
        if (dd == 1)
        {
                six();
        }
        glPopMatrix();

        glPushMatrix();
        if (da == 1)
        {
                seven();
        }
        glPopMatrix();

        glPushMatrix();
        if (db == 1)
        {
                eight();
        }
        glPopMatrix();
```

```
        glutSwapBuffers();
        glFlush();
}
```

- **Keyboard input**

```
void CS2(int o)
{
        if (o == 1)
        {
                count = 1;
                glutIdleFunc(twotwoU);
        }
        if (o == 2)
        {
                count1 = 1;
                glutIdleFunc(twotwoD);
        }

}
void CS1(int op)
{
        if (op == 2)
        {
                glutIdleFunc(oneoneU);

        }
        if (op == 1)
        {
                if (flag == 0)
                {
                        glutIdleFunc(handshake1H);
                }
```

```
        }

        if (op == 3)
        {
                glutIdleFunc(oneoneD);
        }

}

void cs(int OP)
{
        if (OP == 5)
        {
                exit(0);
        }
}
```

- **Encryption process**

```
void encryption()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
         glColor3f(0.25,0.15,0.36); glColor3f(1.0f, 1.0f, 1.0f);
        output(150, 640, "ENCRYPTION AND DECRYPTION PROCESS", 2);
        glPopMatrix();
        glColor3f(1.0f, 1.0f, 1.0f); glPushMatrix(); glTranslatef(-30, 200, 0);
        output(75, 440, "Sender", 2); computer();
        glPopMatrix();
        glColor3f(1.0f, 1.0f, 1.0f); glPushMatrix();
        output(565, 630, "Receiver", 2); computer_dest(); glPopMatrix();
        glColor3f(1.0f, 1.0f, 1.0f); glPushMatrix(); glScalef(70, 40, .5);
        glTranslatef(1.5, 7, 0); glutWireCube(2); glPopMatrix();
```

```
if (phy_header_angle1 != 100)
        {
                glPushMatrix();
                glTranslatef(0, -phy_header_angle1, 0);
                glPushMatrix();
                glTranslatef(0, -arrow_angle, 0); glTranslatef(0, 150, 0); message_data();
                glPopMatrix(); glPushMatrix();
                glTranslatef(phy_header_angle, 0, 0); glTranslatef(-100, 0, 0);
message_key();
                glPopMatrix();
                glPopMatrix();
        }
        glPushMatrix(); glTranslatef(movement_angle1, 0, 0); if (phy_header_angle1 ==
100)
        {
                glPushMatrix();
                if (movement_angle1 >= 420)
                {
                        glRotatef(180,0,1,0); glTranslatef(0, rev_analog_sig_angle, 0);
                }
                glTranslatef(0, -analog_sig_angle, 0); if (rev_analog_sig_angle != 100)
cipher();
                glPopMatrix();
        }
        glPopMatrix(); glPushMatrix();
        glTranslatef(movement_angle1, 0, 0); if (phy_header_angle1 == 100)
        {
                glPushMatrix(); glScalef(50, 30, .5); glTranslatef(2.5, 2.5, 0);
glutWireCube(2); glPopMatrix();
        }
        glPopMatrix(); if (rev_analog_sig_angle == 100)
        {
                glPushMatrix();
                glTranslatef(450, rev_phy_header_angle1, 0); glPushMatrix();
```

```
                glTranslatef(0,rev_arrow_angle,0);
                glTranslatef(0,0,0); message_data(); glPopMatrix(); glPushMatrix();
                glTranslatef(rev_phy_header_angle, 0, 0); glTranslatef(-10, 0, 0);
                if (rev_phy_header_angle < 200)
                        message_key();
                glPopMatrix(); glPopMatrix();
        }
        glColor3f(1.0f, 1.0f, 1.0f); glPushMatrix();glTranslatef(450, 0, 0)   glScalef(70, 40,
.5); glTranslatef(1.5, 7, 0); glutWireCube(2);glPopMatrix();animation_encryp(); glFlush();
        glutSwapBuffers();
```

- **Upload**

```
void upload()
{
        glPushMatrix();
        glTranslatef(100.0, 0.0, 0.0);
        ptr1 = "U P L O A D ";
        ptr2 = "E D ";
        len1 = strlen(ptr1);
        len2 = strlen(ptr2);
        glColor3f(0.60, 0.60, 0.60);
        glRasterPos3f(20.0, 315.0, 0.0);
        for (int i = 0; i < len1; i++)
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, ptr1[i]);
        for (int i = 0; i < len2; i++) {

                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, ptr2[i]);
        }
        glPopMatrix();
        glutPostRedisplay();
        glFlush();
}
```

- **Download**

```
void downloading()// used to display the bitmap characters.
{
	glPushMatrix();
	glTranslatef(100.0, 0.0, 0.0);
	ptr5 = "D O W N L O A D I N G ... ";
	len5 = strlen(ptr5);
	//glColor3f(0.60, 0.60, 0.60);
	glRasterPos3f(15.0, 190.0, 0.0);
	for (int i = 0; i < len5; i++)
		glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, ptr5[i]);

	glPopMatrix();
	glutPostRedisplay();
	glFlush();
}
```

- **Handshake process**

```
void handshake1V()
{
	if (p3 <= 175.0 && p4 == 200.0 && p3 > 100.0)
	{
		p3 = p3 - 0.08;
	glPushMatrix();
		glPointSize(2.0);
		glColor3f(1.0, 1.0, 1.0);
		glBegin(GL_POINTS);
		glVertex2f(p3, p4);
		glEnd();
		glPopMatrix();
		glutPostRedisplay();
	}
	else
	{
		if (p3 <= 100.0 && p4 < 252.0)
```

```
                {
                        p4 = p4 + 0.08;

                        glPushMatrix();
                        glColor3f(1.0, 1.0, 1.0);
                        glPointSize(2.0);
                        glBegin(GL_POINTS);
                        glVertex2f(p3, p4);
                        glEnd();
                        glPopMatrix();
                        glutPostRedisplay();
                }
        }
        glPushMatrix();
        glColor3f(0.6, 0.6, 0.6);
        response();
        glPopMatrix();

        glutPostRedisplay();
        glFlush();
}

void handshake1H()
{
        if (p1 >= 107.0 && p2 == 300.0 && p1 < 200.0)
        {
                p1 = p1 + 0.08;
                glPushMatrix();
                glPointSize(2.0);
                glColor3f(1.0, 1.0, 1.0);
                glBegin(GL_POINTS);
                glVertex2f(p1, p2);
glEnd();
                glPopMatrix();
```

```
                glutPostRedisplay();
        }
        else
        {
                if (p1 >= 200.0 && p2 > 278.0)
                {
                        p2 = p2 - 0.08;

                        glPushMatrix();
                        glColor3f(1.0, 1.0, 1.0);
                        glPointSize(2.0);
                        glBegin(GL_POINTS);
                        glVertex2f(p1, p2);
                        glEnd();
                        glPopMatrix();
                        glutPostRedisplay();
                }
        }
        glPushMatrix();
        glColor3f(0.6, 0.6, 0.6);
        request();
        glPopMatrix();
        if (p1 >= 200.0 & p2 <= 278.0)
        {
                handshake1V();
        }
        flag = 1;
        glFlush();
}
```

Chapter 6

RESULTS

- **Start Page**

  The starting page of the application, giving options to the user to select which page is to be accessed next using keyboard interaction or mouse input.

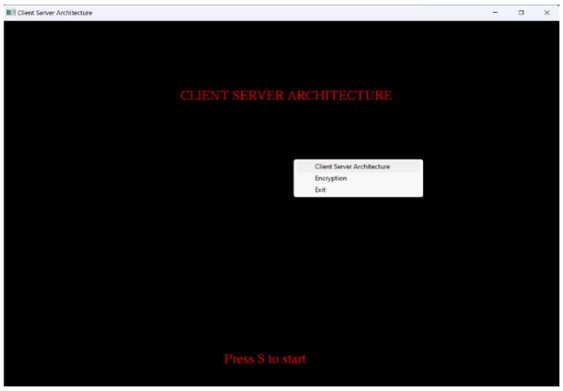

Figure 6.1 Start Page of the Application

- **Encryption and Decryption process**

  Shows the encryption of the message from the sender, followed by the decryption of the same ciphertext back to the original message at the receiver's end.

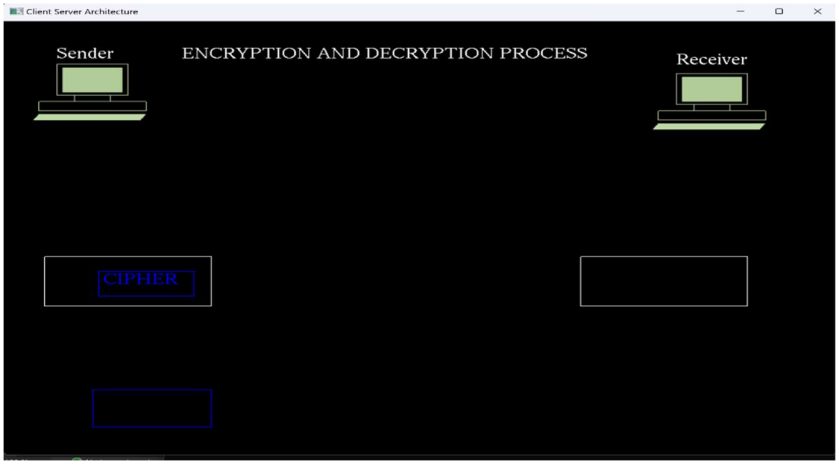

Figure 6.2 Encryption and Decryption process

- **Client Server Architecture Page**

  In this page the user is allowed to choose 2 of the available protocols, TCP or UDP, and upload or download the data from the server to the client computer.

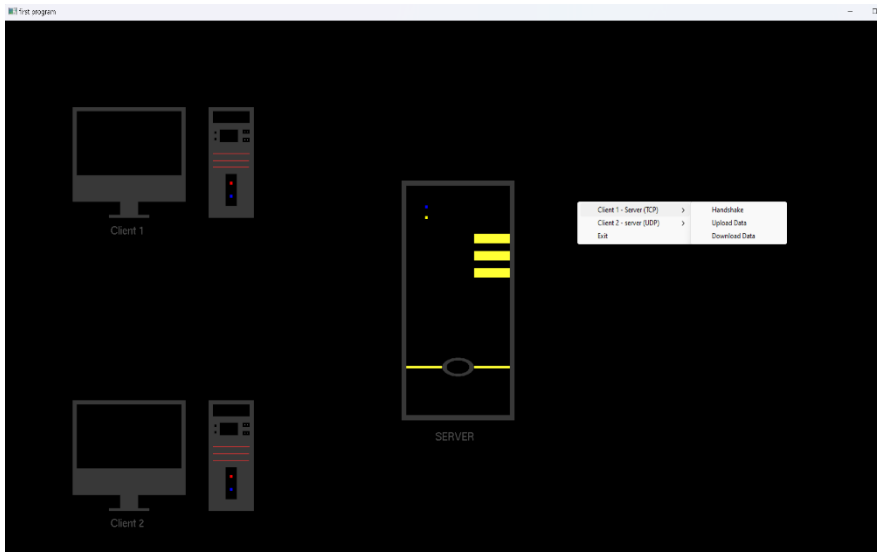

Figure 6.3 Client Server architecture page

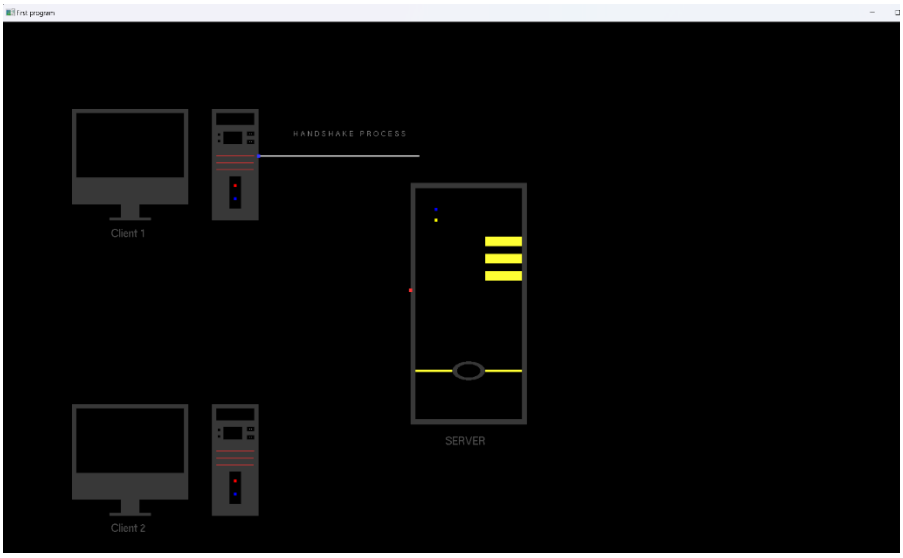

Figure 6.4 Handshake process between client and server

- Download data from server using TCP.

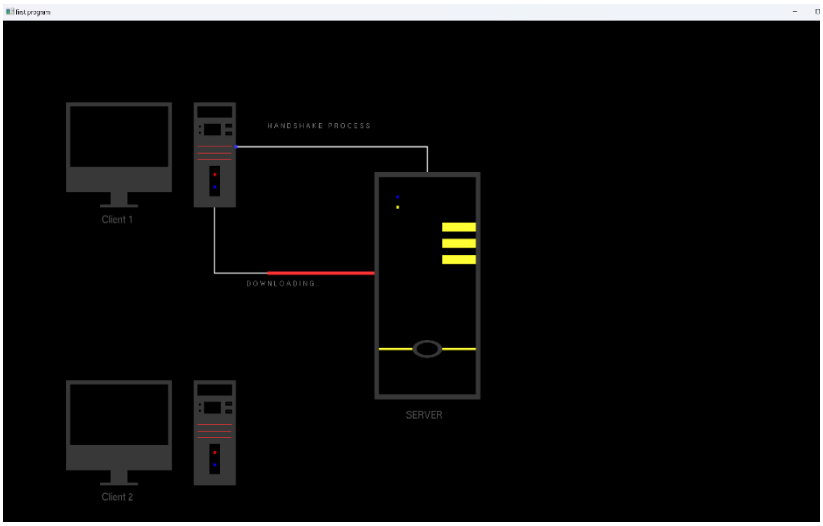

Figure 6.5 Download data from server

- Upload data to server using UDP.

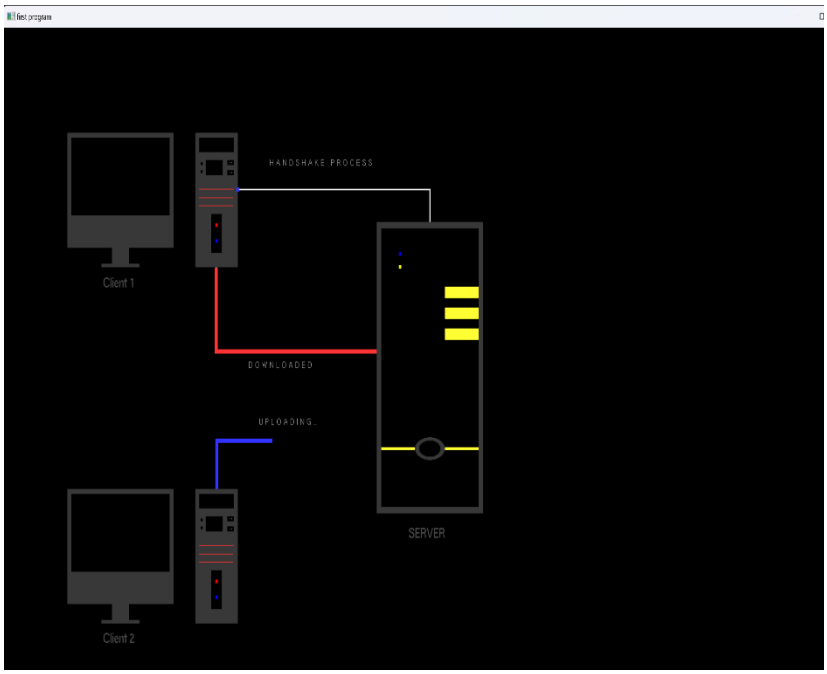

Figure 6.6 Upload data to server

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

The provided code demonstrates a simple OpenGL program that creates a window and handles keyboard input using the GLUT library. It includes a function keyboard to exit the program when the 'q' key or the Escape key is pressed. The main function initializes OpenGL, sets up the window, registers the keyboard input function, and enters the main event loop.

Future Enhancements:

- **Input Handling:** Extend the keyboard function to handle more keys or key combinations for different actions within the program.
- **Graphics Ryendering:** Enhance the program to render more complex graphics or animations based on user input.
- **User Interface:** Implement a more interactive user interface with menus, buttons, and other GUI elements that respond to keyboard input.
- **Error Handling:** Implement error handling mechanisms to gracefully handle unexpected errors or invalid input.
- **Optimization:** Optimize the rendering process and resource usage for better performance, especially for more complex scenes or animations.
- **Platform Compatibility:** Ensure the program works seamlessly across different platforms and OpenGL implementations by testing and making necessary adjustments.
- **Documentation:** Improve code documentation to make it more understandable and maintainable for future development or collaboration.
- **User Experience:** Enhance the user experience by providing feedback or visual cues in response to keyboard input actions.

BIBLIOGRAPHY

1. https://www.geeksforgeeks.org/difference-between-encryption-and-decryption/

2. https://www.geeksforgeeks.org/client-server-model/

3. https://www.pkware.com/blog/client-side-encryption-vs-end-to-end-encryption-whats-the-difference

4. https://www.outsystems.com/forums/discussion/86040/client-side-encrypt-and-decrypt-server-side/