# VISVESVARAYA TECHNOLOGICAL UNIVERSITY
## BELAGAVI, KARNATAKA-590 018.

## A MINI PROJECT REPORT
**ON**
# ANALOG CLOCK

**Submitted in partial fulfilment of the requirements for the Mini Project (18CSL67) course of the 6th semester.**

**BACHELOR OF ENGINEERING IN**

**COMPUTER SCIENCE AND ENGINEERING**

**By**


**HARSH MISHRA (1JS20CS068)**

**KARUNA SAGAR (1JS20CS077)**


**Under the guidance of**

**Dr. Pavithra G S**

**Asst. Professor, Dept. of CSE**


**JSS MAHAVIDYAPEETHA, MYSURU**

# JSS Academy of Technical Education

**JSS Campus, Uttarahalli Kengeri Main Road, Bengaluru – 560060**

## Department of Computer Science and Engineering


# CERTIFICATE

**This is to certify that the mini project work entitled "ANALOG CLOCK" is a benefited work carried out by HARSH MISHRA bearing USN 1JS20CS068, KARUNA SAGAR bearing USN 1JS20CS077 bonafide student of JSS Academy of Technical Education in the partial fulfillment for the award of the Bachelor of Engineering in Computer Science & Engineering of the Visvesvaraya Technological University, Belgaum, during the year 2022-23. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of mini-Project work prescribed for the said degree.**

**Dr. Pavithra G S**
**Asst. Professor, Dept. of CSE**

**Dr. P B Mallikarjuna**
**Assoc. Prof & HOD, Dept. of CSE**

| Name of the Examiners | Signature with date |
|---|---|
| 1. ..... | ..... |
| 2. ..... | ..... |

# ACKNOWLEDGEMENT

**I, take this opportunity to thank one and all involved in helping me build this project. Firstly, I would like to thank the college for providing me an opportunity to work on this project.**

**I thank the management of the JSS Academy of Technical Education for providing all the resources required for the project.**

**I wish to acknowledge my sincere gratitude to our Principal, Dr. Bhimasen Soragaon for his constant encouragement and for providing us with all the facilities required for the accomplishment of this project.**

**The project would not have been possible if not for the constant support of our Associate Professor and Head of the Computer Science Department, Dr. Mallikarjuna P B.**

**I also am highly grateful to the guidance offered by Dr. Pavithra G S, Asst. Professor, Department of Computer Science, who has been very generous in assisting and supporting, to do this project named "ANALOG CLOCK", which formally started as just a rough idea and now has resulted in the form of this project.**

**I also would like to thank all the other teaching and non-teaching staff members who had extended their hand for support and co-operation while bringing up this project.**

**HARSH MISHRA(1JS20CS068)**
**KARUNA SAGAR(1JS20CS077)**

# ABSTRACT

**This project aims to develop an analog clock using the C++ programming language and the OpenGL graphics library. The clock is designed to display the current time using hour, minute, and second hands on a circular dial. The project utilizes various concepts of computer graphics and visualization to create an interactive and visually appealing clock.**

**The analog clock is implemented using OpenGL primitives and functions. It features a circular dial with Roman numerals or digits indicating the hours. The hour, minute, and second hands rotate dynamically based on the current time. The clock's appearance can be customized by selecting different colors, such as red, green, or blue.**

**The program also includes additional functionalities. It initially displays a welcome page with information about the project, including the names of the developers and the project's supervisor. The clock can be started by pressing the spacebar, which transitions from the welcome page to the clock display. Additionally, a right-click menu allows the user to change the color of the clock and toggle between Roman numerals and digits for the hour markers.**

**The analog clock project demonstrates the implementation of a graphical application using OpenGL and C++. It combines elements of computer graphics, interactive user interfaces, and basic game development concepts. By exploring this project, students can gain insights into computer graphics programming, visual representation of data, and user interaction with graphical.**

# CONTENTS

# LIST OF FIGURES

# Chapter 1: Introduction:

The ANALOG CLOCK is a computer graphics and visualization project that utilizes the power of C++ programming language and the OpenGL graphics library to create an immersive gaming experience. In this section, we will provide an overview of the project and its objectives.

The main goal of the project is to develop an engaging analog clock using OpenGL graphics library. The clock will provide an immersive experience where users can interact with various elements and visualize the passage of time. The clock's design will be visually appealing, incorporating aesthetic backgrounds and captivating visual effects. The use of OpenGL will enable realistic rendering of clock hands and smooth animation transitions. By exploring the principles of computer graphics and visualization, this project aims to create a captivating and interactive analog clock experience for users.


**FIG:1.1 OVERVIEW OF COMPUTER GRAPHICS**

## 1.1 About OpenGL:

OpenGL (Open Graphics Library) is a cross-platform graphics API (Application Programming Interface) that allows developers to create interactive and highperformance 2D and 3D graphics applications. It provides a set of functions for rendering and manipulating graphical objects, enabling developers to harness the full potential of modern graphics hardware.

## 1.2.1 OpenGL Commands and Primitives:

OpenGL provides a wide range of commands and primitives that developers can use to create and manipulate graphical objects. Primitives are basic geometric shapes such as points, lines, and polygons. These primitives can be combined and transformed to create complex objects in the game environment.

OpenGL commands allow developers to control the rendering process, specifying various attributes such as color, texture, and lighting effects. For example, developers can use commands to set the background color, define the position and orientation of objects, and apply textures to surfaces.

## 1.2.2 OpenGL Rendering Pipeline:

The OpenGL rendering pipeline is a series of stages that transform the input data (vertices, textures, etc.) into the final image displayed on the screen. Understanding the rendering pipeline is crucial for efficient graphics programming. The pipeline consists of several stages, including vertex processing, primitive assembly, rasterization, fragment processing, and framebuffer operations. Each stage performs specific tasks such as transforming vertices, interpolating colors, applying textures, and performing depth testing.

By properly understanding and utilizing the rendering pipeline, developers can optimize their code and achieve better performance in rendering complex scenes.

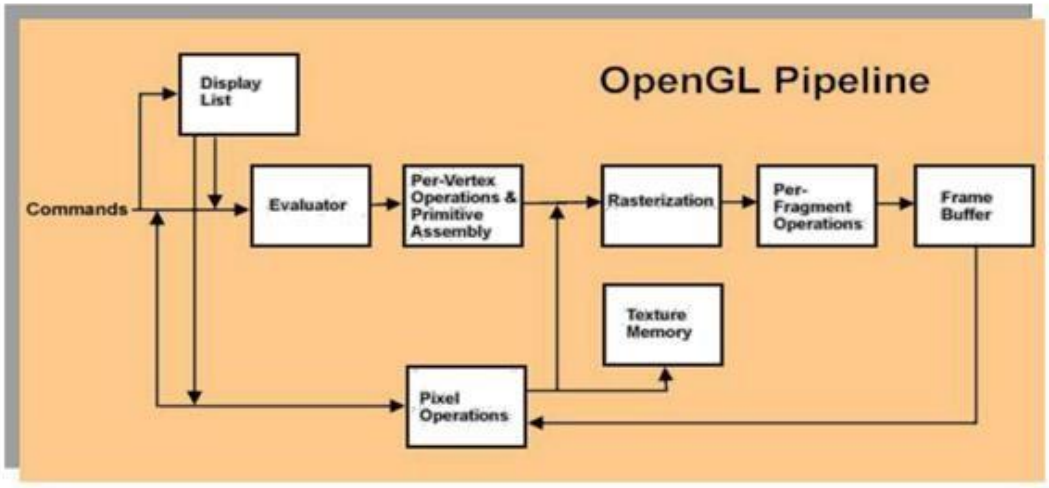



**FIG:1.2.2 OPENGL PIPELINE ARCHITECTURE**

## 1.2.3 OpenGL - GLUT and OpenGL Utility Libraries:

GLUT (OpenGL Utility Toolkit) and OpenGL Utility Libraries are additional tools and libraries that provide extended functionality and convenience for OpenGL programming. GLUT simplifies the process of creating windows, handling user input, and managing events such as mouse clicks and keyboard inputs. It provides a platform-independent interface for interacting with the underlying operating system, allowing developers to focus on graphics programming without worrying about platform-specific details.

In the ANALOG CLOCK project, we will leverage the power of GLUT and OpenGL Utility Libraries to streamline the development process and create a user-friendly interface for controlling the UFO spacecraft and interacting with the game world.

OpenGL Utility Libraries, such as GLU (OpenGL Utility Library) and GLM (OpenGL Mathematics), offer additional utilities and functions for common tasks like loading 3D models, handling matrix transformations, and performing mathematical operations. These libraries enhance the development process and facilitate efficient and robust code implementation.

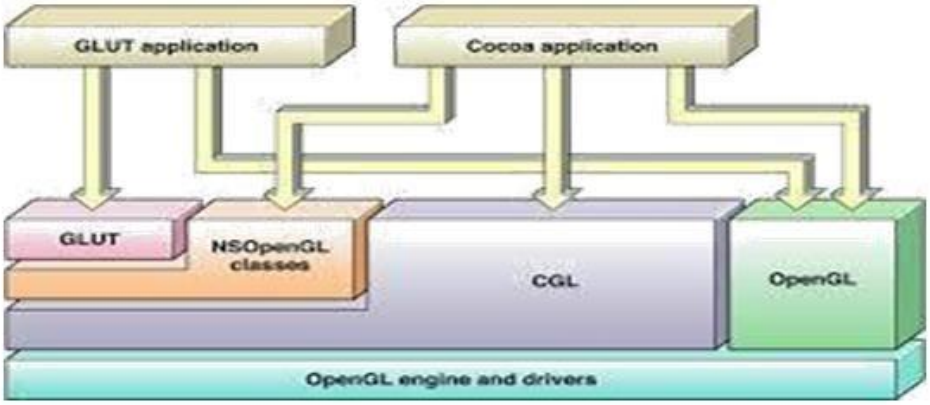



**FIG:1.2.2 OPENGL PIPELINE ARCHITECTURE**

By utilizing OpenGL, along with its commands, rendering pipeline, and utility libraries, the ANALOG CLOCK project aims to provide an immersive gaming experience with visually appealing graphics, smooth animations, and realistic physics-based movements.

This project offers an excellent opportunity for students to explore the fundamentals of computer graphics and visualization while developing their programming skills in C++ and OpenGL.

## 1.3 Advantages of Using OpenGL:

OpenGL offers several advantages that make it a popular choice for computer graphics programming:

Cross-Platform Compatibility: One of the significant advantages of OpenGL is its cross-platform compatibility. It is supported on various operating systems, including Windows, macOS, and Linux, allowing developers to create applications that can run seamlessly across different platforms without extensive modifications.

Hardware Acceleration: OpenGL takes advantage of the capabilities of modern graphics hardware to accelerate the rendering process. By offloading computations to the GPU (Graphics Processing Unit), OpenGL can achieve faster rendering and deliver high-performance graphics, making it ideal for real-time applications like games.

Wide Industry Adoption: OpenGL has been widely adopted in various industries, including gaming, virtual reality, scientific visualization, and computer-aided design. Its extensive use in the industry ensures a wealth of resources, tutorials, and community support, making it easier for developers to find help and learn from others' experiences.

Flexibility and Customization: OpenGL provides developers with a high degree of flexibility and customization options. It allows fine-grained control over the rendering pipeline, enabling developers to optimize their code for specific requirements. Additionally, OpenGL supports extensions that allow developers to access advanced graphics features and tailor their applications to specific hardware capabilities.

Integration with Other Libraries: OpenGL can be easily integrated with other libraries and frameworks to enhance functionality. For example, developers can combine OpenGL with

libraries like OpenAL for audio, OpenCL for parallel computing, or ImGui for **creating** graphical user interfaces. This flexibility allows developers to leverage the strengths of different libraries and create powerful applications.

## 1.4 C++ and OpenGL:

C++ is a versatile and widely used programming language known for its performance, efficiency, and extensive support for object-oriented programming. When combined with OpenGL, a powerful graphics library, it becomes a formidable tool for computer graphics programming. In this section, we will explore the benefits and capabilities of using C++ in conjunction with OpenGL.

C++ supports object-oriented programming paradigms, enabling developers to design modular and reusable code structures. This is advantageous in graphics programming, as it allows for the creation of classes and objects that encapsulate graphics entities, such as meshes, textures, and shaders. Object-oriented design promotes code organization, readability, and maintainability, making it easier to develop and maintain large-scale graphics applications.

### 1.4.1 Performance and Efficiency:

C++ is renowned for its ability to deliver high-performance code. Its low-level control and direct memory management allow developers to optimize their programs for efficiency. This is particularly important in graphics programming, where real-time rendering and complex computations are required. With C++, developers have finegrained control over memory allocation, data structures, and algorithm implementations, resulting in faster and more efficient code execution.

### 1.4.2 Object-Oriented Programming:

C++ supports object-oriented programming paradigms, enabling developers to design modular and reusable code structures. This is advantageous in graphics programming, as it allows for

the creation of classes and objects that encapsulate graphics entities, such as meshes, textures, and shaders. Object-oriented design promotes code organization, readability, and maintainability, making it easier to develop and maintain large-scale graphics applications.

### 1.4.3 Compatibility and Portability:

C++ is a widely supported language, with compilers available for various platforms and operating systems. This ensures that C++ code written for graphics programming using OpenGL can be easily ported to different systems without significant modifications. This cross-platform compatibility enables developers to reach a broader audience and deploy their applications on multiple platforms, such as Windows, macOS, and Linux.

### 1.4.4. Leveraging the Features of OpenGL:

Graphics Rendering: OpenGL provides a comprehensive set of functions and features for rendering 2D and 3D graphics. It supports a variety of rendering primitives, including points, lines, and polygons, which can be transformed and textured to create complex objects. OpenGL also offers advanced rendering techniques such as vertex and fragment shaders, which allow developers to manipulate vertices and fragments at the GPU level, achieving stunning visual effects.

Platform-Independent API: OpenGL is designed to be a platform-independent graphics API. It provides a consistent interface across different operating systems, allowing developers to write graphics code that can be executed on multiple platforms without modification. By utilizing OpenGL in C++, developers can write graphics code that is portable and can seamlessly run on various systems, ensuring broad compatibility and reach.

Extensibility and Compatibility: OpenGL is an extensible API that supports the use of extensions. These extensions provide additional features and functionalities beyond the core OpenGL specification. This extensibility allows developers to leverage the latest graphics capabilities and hardware advancements. Furthermore, OpenGL maintains backward

compatibility, ensuring that applications written using older versions of the API can still run on newer systems with minimal adjustments.

Libraries and Frameworks: C++ and OpenGL benefit from a vast array of libraries and frameworks that facilitate graphics programming. These include GLM (OpenGL Mathematics) for vector and matrix operations, GLFW (Graphics Library Framework) for window creation and user input handling, and Assimp for loading and processing 3D model files. These libraries enhance development productivity by providing pre-built functionality and simplifying common tasks.

IDEs and Debugging Tools: C++ is supported by numerous Integrated Development Environments (IDEs) that offer powerful features for code editing, debugging, and profiling. Popular IDEs such as Visual Studio, CLion, and Code::Blocks provide comprehensive development environments tailored for C++ programming. Additionally, debugging tools like GDB (GNU Debugger) enable developers to identify and fix errors efficiently during the development process.

C++ and OpenGL have thriving communities of developers, enthusiasts, and experts who actively contribute to online forums, discussion boards, and tutorial websites. These communities provide a wealth of knowledge, code samples, and troubleshooting advice, making it easier for developers to learn, solve problems, and stay up-to-date with the latest advancements in graphics programming.

In conclusion, combining C++ with OpenGL offers a powerful and flexible approach to graphics programming. C++ provides performance, efficiency, and compatibility, while OpenGL delivers a comprehensive set of graphics rendering capabilities. Together, they form a formidable duo that allows developers to create visually stunning and highperformance graphics applications across multiple platforms. By leveraging the features of both C++ and OpenGL, developers can unlock the full potential of computer graphics and bring their visions to life.

## 1.5 Future of OpenGL:

While OpenGL has been a dominant graphics API for many years, its future is evolving with the emergence of new technologies and APIs. The Khronos Group, the organization **responsible for** OpenGL's development, has introduced Vulkan as a next-generation graphics API designed to provide even greater performance and efficiency.

Vulkan, also known as OpenGL Next or OpenGL 4.6+, offers lower-level access to graphics hardware, allowing developers to have more control over the rendering process and achieve higher performance. Vulkan aims to address the limitations of OpenGL and provide a more modern and efficient graphics API.

However, despite the rise of Vulkan, OpenGL remains relevant and widely used, especially in legacy systems and applications. Many existing applications, including games, continue to rely on OpenGL, and support for the API is expected to continue for the foreseeable future.

Furthermore, OpenGL continues to evolve, with periodic updates and extensions being introduced to add new features and improve functionality. Developers who are familiar with OpenGL can easily transition to Vulkan or other APIs as needed, leveraging their existing knowledge and experience.

In conclusion, while the future of graphics programming may be shifting towards newer APIs like Vulkan, OpenGL remains a powerful and widely adopted graphics library with cross-platform compatibility, hardware acceleration, and a flexible development environment. Its ease of use, extensive resources, and broad industry support make it an excellent choice for the ANALOG CLOCK project, providing a solid foundation for creating captivating and visually stunning graphics.

The below diagram illustrates the relationship of the various libraries and window system components.

Generally, applications which require more user interface support will use a library designed to support those types of features (i.e. buttons, menu and scroll bars, etc.) such as Motif or the Win32 API.

Prototype applications, or ones which do not require all the bells and whistles of a full GUI, may choose to use GLUT instead because of its simplified programming model and window system independence**.**

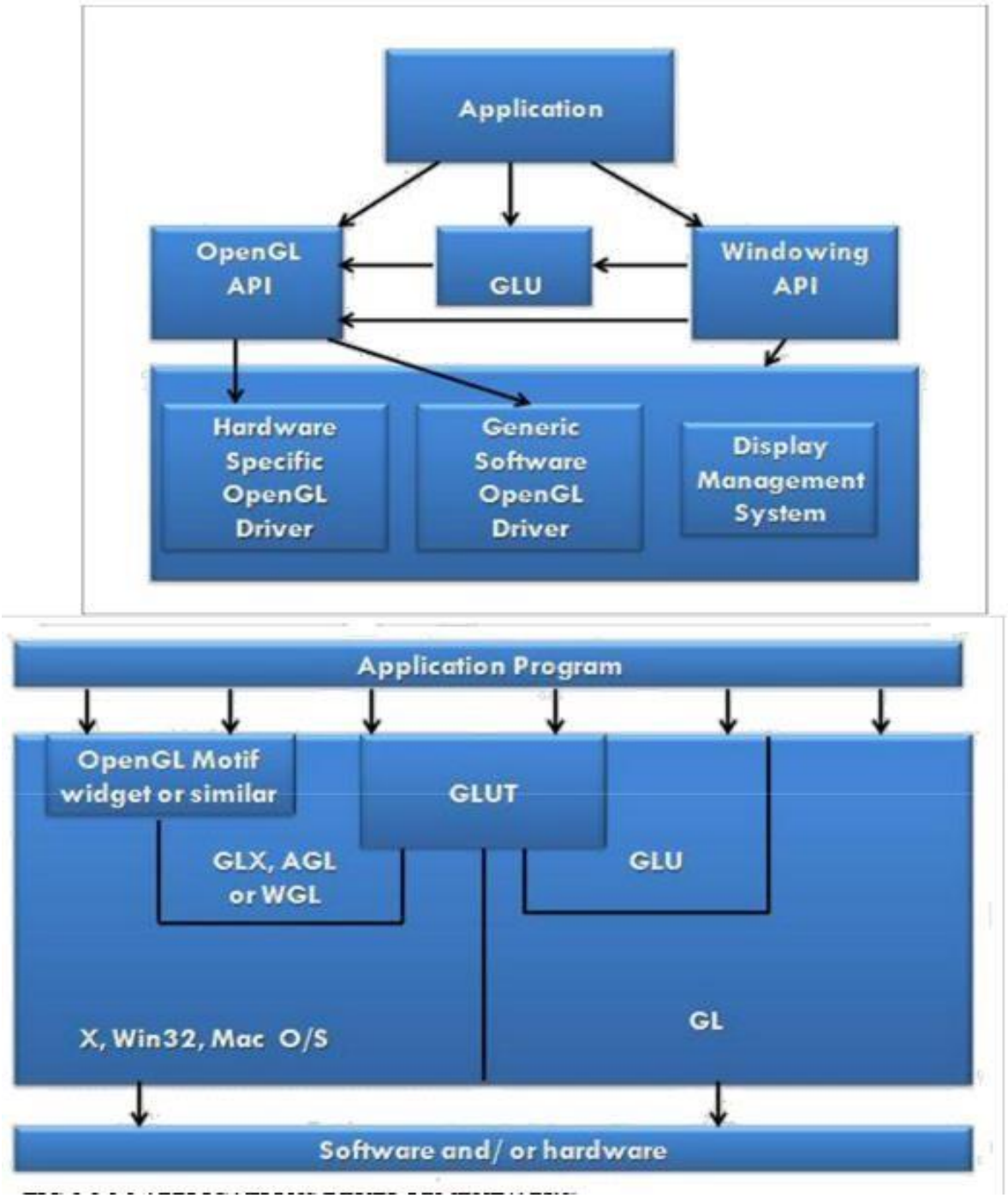



**FIG:1.5 APPLICATIONS DEVELOPMENT(API'S)**

# Chapter 2: Requirement Specifications

## 2.1 SOFTWARE SPECIFICATION

Operating System: Windows 11 and Linux Mint

Libraries: OpenGL Libraries

IDE: Code::Blocks

## 2.2 HARDWARE SPECIFICATION

Processor: x86 compatible processor with 1.7 GHz Clock Speed

RAM: 512 MB or greater

Hard Disk: 20 GB or grater

Monitor: VGA/SVGA

Keyboard: 104 keys standard

Mouse: 2/3 button. Optical/Mechanical.

## 2.3 USER CHARACTERISTICS

Every user:

Should be comfortable with basic working of the computer Must have basic knowledge of English

Must carry a login ID and password used for authentication

# Chapter 3: Design Phase

- Here are the steps to develop the Analog Clock program:
- Initialize the necessary libraries and variables.
- Define constants for the screen size, clock parameters, and other necessary values.
- Create arrays to store the coordinates of various shapes used in the clock.
- Implement a function to display raster text on the screen.
- Create an initialization function to set up the OpenGL environment.
- Implement a function to display the introduction screen with the clock details and project information.
- Implement a function to display the start screen with options to start the clock, view instructions, or quit.
- Implement a function to handle the back button in the instructions screen.
- Implement a function to display the instructions screen with clock instructions for the users.
- Implement functions to draw the different parts of the clock, such as the hour hand, minute hand, and second hand.
- Implement a function to draw the clock's numbers and markers.
- Set up the main display function to handle different view pages and call appropriate functions accordingly.
- Implement keyboard and mouse input handling functions to interact with the clock.
- Set up the main function to initialize the OpenGL environment, register callback functions, and enter the main loop.
- Please note that the above steps outline the general process involved in developing the Analog clock program. You would need to write the actual code to implement each step according to your specific requirements and programming language.

## 3.1 Program Structure :

### clockLayout() :

I've used this function to print the clock layout i.e. clock dial and the markings on the clock. If we observe clearly, the clock has hours marking each separated

by 30 degrees and each hour is divided into 5 markings each making an angle of 6 degrees. So, iterating the markings for every 30 degrees.

## 3.2 Image Theory

### secHand()

It is clear from the name that this gonna do something with the seconds hand. This function is going to get the present second from the system clock and incline the line according to a particular angle. Eg: if the present seconds is 5 then the angle of the seconds hand with respect to the vertical must be 30 degrees, i.e. 5*6=30**.**

### minHand()

This function fulfills the task of moving the minutes hand based on the system clock. The minutes hand must be inclined 6 degrees for every minute passing. Eg: if the elapsed minutes are 30 then the minutes hand angle must be making 180 degrees with the vertical.

### hrHand()

This function is going to print an inclined hours line. The function is designed to get the present hour and also the no. of elapsed minutes from the system clock and incline the line according to a particular angle. For every hour elapsed the hour hand moves 30 degrees and every 12 minutes it moves 6 degrees.

### main()

The first lines in main are graphic initialization, you must change the path

“c:\turboc3\bgi\” to your compiler’s BGI file path otherwise program will not work. Coming to the while loop, the while loop iterates for every 100 milliseconds reprinting all the functions. This program is like getting the static picture of clock every second and combining all the pictures to make a moving analog clock.

## 3.3 Working of Analog Clock

Clocks use oscillators to keep the gears in motion. These oscillators have changed from flowing water to pendulums to quartz crystals, but they all seek to achieve the same purpose: perpetual motion. The oscillating mechanisms are powered by a variety of sources, but the two most prevalent power sources are direct electrical currents or batteries. The electricity or battery power fuels the controller mechanism, which supplies crucial support to the oscillator. The controller, in turn, fuels the wheel chain of the analog clock. The wheel chain is responsible for notching and turning the indicator hands of the clock. So, with each second that passes, the wheel chain synchronizes the minutes and hours accordingly. It is a relatively simple process, culminating in the familiar big hand and little hand indicating the hour and minute. All analog clocks uses gears in intricate fashion to enhance efficiency but the basic mechanism remains same (even for vertical designs)..

## 3.4 Program Graphics

Most of the objects in the program, such as buttons, grids or cells, are drawn using OpenGL primitives.

Most of the control buttons used through the program were made as textures in an image editor and exported as .raw files. The class *"Textures"* (in *Textures.h* & *Textures.cpp*) handles the loading of these images intomemory and drawing them on the screen using OpenGL's texture related function.

Text is drawn after the backbuffer is flushed to the front buffer, by hijacking the window handle wia Win32 and writing text onto the window via the Win32
ThrowText() function defined in *Util.h*

# 4. IMPLEMENTATION SOURCE CODE

```
#include <iostream>
#include <GL/gl.h>
#include <GL/glut.h>
#include <GL/freeglut.h>
#include <math.h>
#include <time.h>
#include <string>
#include <cstring>

const GLfloat tam_x = 50.0f; const
GLfloat tam_y = 50.0f;

const GLint sy = 30; const
GLint my = 25; const
GLint hy = 20;

int hour; int minute; int second; int currentColor =
0; // 0: Red, 1: Green, 2: Blue
bool useRomanNumerals = false; // Flag to determine whether to use Roman numerals or digits
bool showClock = false; // Flag to determine whether to show the clock or the welcome page

void RenderString(float x, float y, void* font, const unsigned char* str)
{    char* c;    glColor3ub(0, 0, 0); // Black color
for numbers    glRasterPos2f(x, y);
glutBitmapString(font, str);
}

void circle(GLfloat xc, GLfloat yc, GLfloat raio, bool fill)
{
```

```
   const GLfloat c = 3.14169f / 180.0f;
   GLint i;

   glBegin(fill ? GL_TRIANGLE_FAN : GL_LINE_LOOP);

   for (i = 0; i <= 360; i += 2)
   {        float a = i * c;        glVertex2f(xc + sin(a) *
raio, yc + cos(a) * raio);
   }

   glEnd();
}

void drawDigits()
{
   unsigned char time_1[4] = "1";     unsigned
char time_2[4] = "2";     unsigned char
time_3[4] = "3";     unsigned char time_4[4] =
"4";     unsigned char time_5[4] = "5";
unsigned char time_6[4] = "6";     unsigned
char time_7[4] = "7";     unsigned char
time_8[4] = "8";     unsigned char time_9[4] =
"9";     unsigned char time_10[4] = "10";
unsigned char time_11[4] = "11";     unsigned
char time_12[4] = "12";

   RenderString(-2, 40, GLUT_BITMAP_TIMES_ROMAN_24, time_12);
   RenderString(0, -40, GLUT_BITMAP_TIMES_ROMAN_24, time_6);
   RenderString(40, 0, GLUT_BITMAP_TIMES_ROMAN_24, time_3);
   RenderString(-40, 0, GLUT_BITMAP_TIMES_ROMAN_24, time_9);
```

```
   RenderString(-35, 20, GLUT_BITMAP_TIMES_ROMAN_24, time_10);
   RenderString(35, 20, GLUT_BITMAP_TIMES_ROMAN_24, time_2);
   RenderString(35, -20, GLUT_BITMAP_TIMES_ROMAN_24, time_4);
   RenderString(-35, -20, GLUT_BITMAP_TIMES_ROMAN_24, time_8);

   RenderString(-20, 35, GLUT_BITMAP_TIMES_ROMAN_24, time_11);
   RenderString(20, 35, GLUT_BITMAP_TIMES_ROMAN_24, time_1);
   RenderString(20, -35, GLUT_BITMAP_TIMES_ROMAN_24, time_5);
   RenderString(-20, -35, GLUT_BITMAP_TIMES_ROMAN_24, time_7);
}

void drawRomanNumerals()
{
   const char* romanNumerals[12] = {       "I",
"II", "III", "IV", "V", "VI",
      "VII", "VIII", "IX", "X", "XI", "XII"
   };

   const GLfloat distanceFromBoundary = 15.0f; // Adjust this value to increase/decrease the
distance

   for (int i = 0; i < 12; i++)
   {
      float angle = i * 30;
      glPushMatrix();
      glRotatef(angle, 0.0f, 0.0f, 1.0f);
      RenderString(0,  tam_x  -          distanceFromBoundary,
GLUT_BITMAP_TIMES_ROMAN_24,      reinterpret_cast<const          unsigned
char*>(romanNumerals[i]));        glPopMatrix();
   }
}
```

```
void drawWelcomePage()
{
   //  Background        glClearColor(0.88f,  0.94f,  0.95f,
1.0f); // Light blue
   glClear(GL_COLOR_BUFFER_BIT);

   // Title
   glColor3ub(70, 70, 70); // Dark gray
   RenderString(-15,  10,  GLUT_BITMAP_HELVETICA_18,  reinterpret_cast<const  unsigned
char*>("ANALOG CLOCK"));

   // Names and Details glColor3ub(0, 0,
0); // Black
RenderString(-20,      -5,      GLUT_BITMAP_HELVETICA_12,          reinterpret_cast<const
unsigned char*>("NAME: HARSH MISHRA    USN:1JS20CS068"));
RenderString(-20,      -10, GLUT_BITMAP_HELVETICA_12, reinterpret_cast<const
unsigned char*>("NAME: KARUNA SAGAR    USN: 1JS20CS077")); glColor3ub(0, 0, 0);
// Black
RenderString(-20,      -15,  GLUT_BITMAP_HELVETICA_12,  reinterpret_cast<const  unsigned
char*>("UNDER THE GUIDANCE OF DR.PAVITHRA GS"));

   // Instructions     glColor3ub(100, 100,
100); // Dark gray     RenderString(-30, -30,
GLUT_BITMAP_HELVETICA_12,
reinterpret_cast<const unsigned
char*>("Press Spacebar to Start the
Clock"));
   glFlush();
}
void draw(void)
```

```
{                    if
(showClock)
   {
      glClear(GL_COLOR_BUFFER_BIT);

      // Color selection based on currentColor
if (currentColor == 0)           glColor3ub(249,
168, 37); // Red       else if (currentColor == 1)
glColor3ub(144, 238, 144); // Light Green
else if (currentColor == 2)       glColor3ub(173,
216, 230); // Light Blue



      circle(0, 0, tam_x, true);

      // Outer boundary
glColor3ub(0, 0, 0); // Black
circle(0, 0, tam_x - 0.5f, false);

      // Inner boundary
glColor3ub(255, 255, 255); // White
circle(0, 0, tam_x - 1.5f, false);

      if (useRomanNumerals)
         drawRomanNumerals();
      else
drawDigits();

      // Second        float angle_s
= second * 6;
```

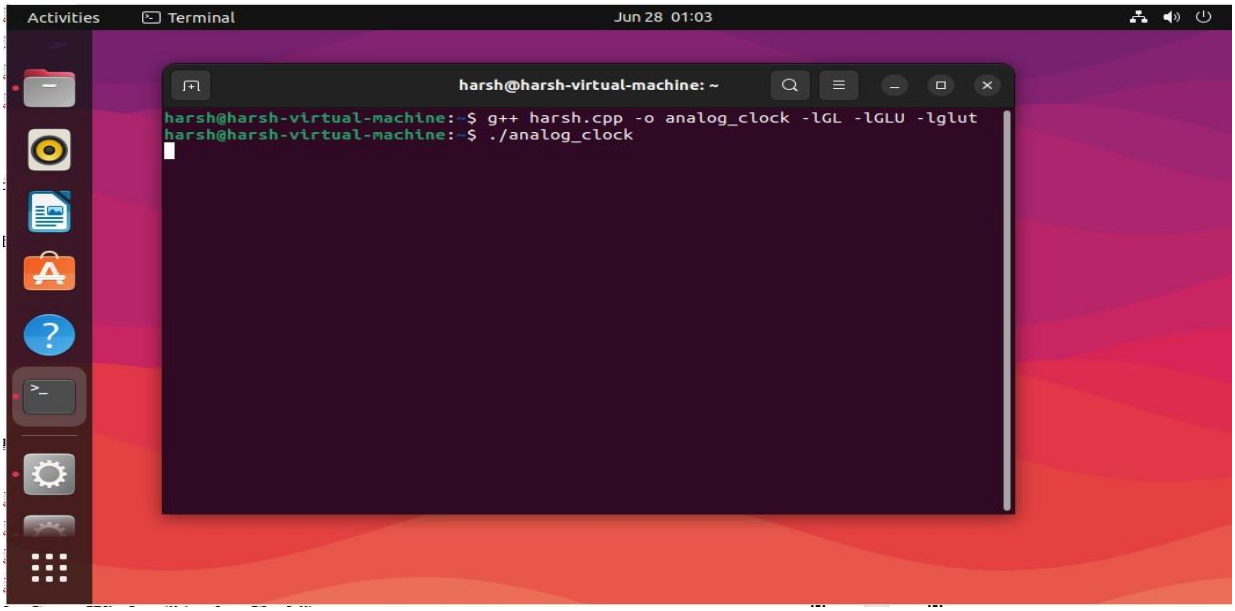
```
        glRotatef(-angle_s, 0.0f, 0.0f, 1.0f);

        glBegin(GL_LINES);
glColor3f(1.0f, 0.0f, 0.0f); // Red
glVertex2i(0, 0);         glVertex2i(0,
sy);         glEnd();

        glLoadIdentity();

        // Minute          float angle_m
= minute * 6;

        glRotatef(-angle_m, 0.0f, 0.0f, 1.0f);

        glLineWidth(3); // Thinner line width
glBegin(GL_LINES);         glColor3f(0.0f,
1.0f, 0.0f); // Green          glVertex2i(0, 0);
glVertex2i(0, my);         glEnd();

        glLoadIdentity();

        // Hour
        float angle_h = (hour + minute / 60.0) * 30;

        glRotatef(-angle_h, 0.0f, 0.0f, 1.0f);

        glBegin(GL_LINES);
glColor3f(0.0f, 0.0f, 1.0f); // Blue
glVertex2i(0, 0);         glVertex2i(0,
hy);         glEnd();
```

```
        glLoadIdentity();

        glFlush();
    }
else
    {
        drawWelcomePage();
    }
}

void changeColor(int option)
{
    currentColor = option;     glutPostRedisplay();
}

void toggleNumerals(int option)
{
    useRomanNumerals = !useRomanNumerals;     glutPostRedisplay();
}

void createMenu()
{
    int colorMenu = glutCreateMenu(changeColor);
glutAddMenuEntry("Red", 0);     glutAddMenuEntry("Green", 1);
glutAddMenuEntry("Blue", 2);

    int numeralMenu = glutCreateMenu(toggleNumerals);        glutAddMenuEntry("Toggle
Numerals", 0);
```

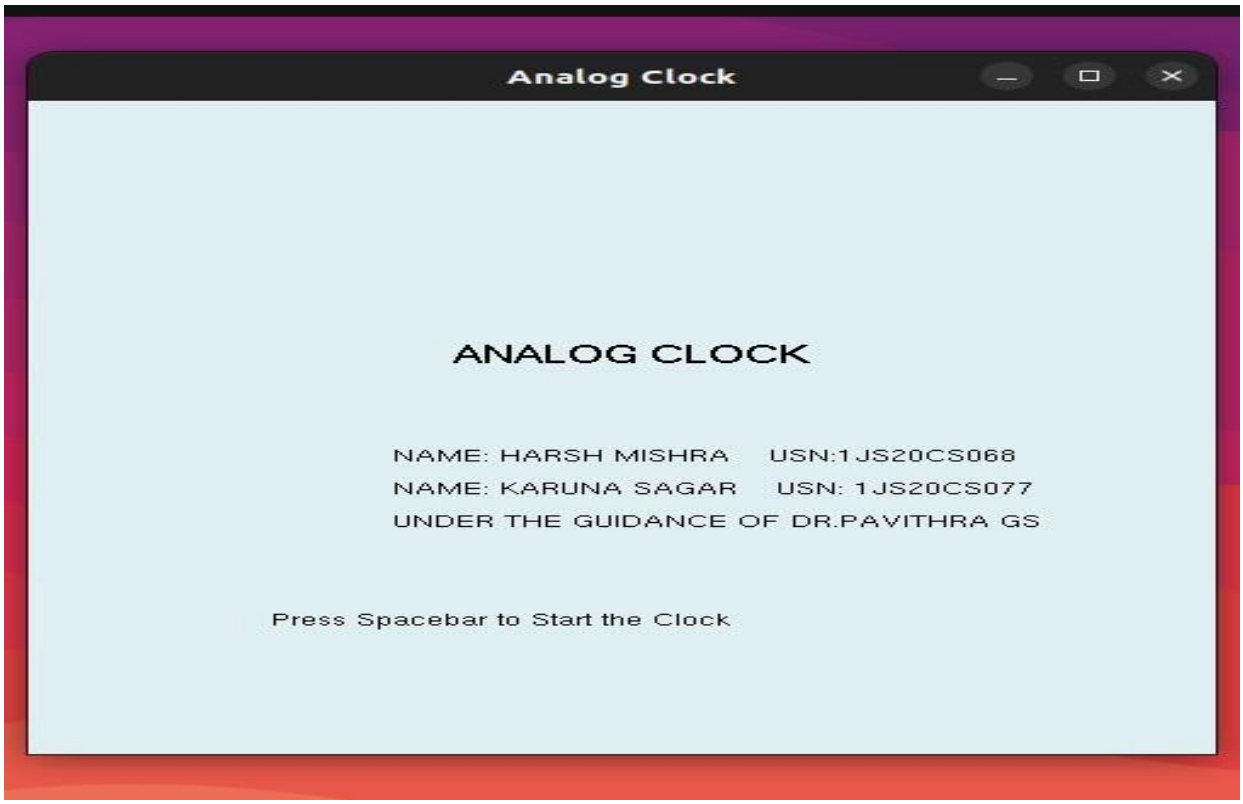
```
   glutCreateMenu(changeColor);     glutAddSubMenu("Change
Color", colorMenu);     glutAddSubMenu("Options",
numeralMenu);     glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void updateClock(int value)
{
   time_t  currentTime  =  time(nullptr);        tm*
localTime = localtime(&currentTime);

   hour = localTime->tm_hour;     minute
= localTime->tm_min;     second =
localTime->tm_sec;

   glutPostRedisplay();     glutTimerFunc(1000,
updateClock, 0);
}

void reshape(int w, int h)
{     glViewport(0, 0, w,
h);
glMatrixMode(GL_PROJ
ECTION);
   glLoadIdentity();     gluOrtho2D(-tam_x,
tam_x, -tam_y, tam_y);
glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{     if (key == 32) // Spacebar key
```

```
   {
      showClock = !showClock;
glutPostRedisplay();
   }
}

int main(int argc, char** argv)
{    glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);
glutCreateWindow("Analog Clock");
glutDisplayFunc(draw);    glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);    createMenu();
updateClock(0);    glutMainLoop();    return 0;
}
```

# Chapter 5: Test Cases and Snapshots



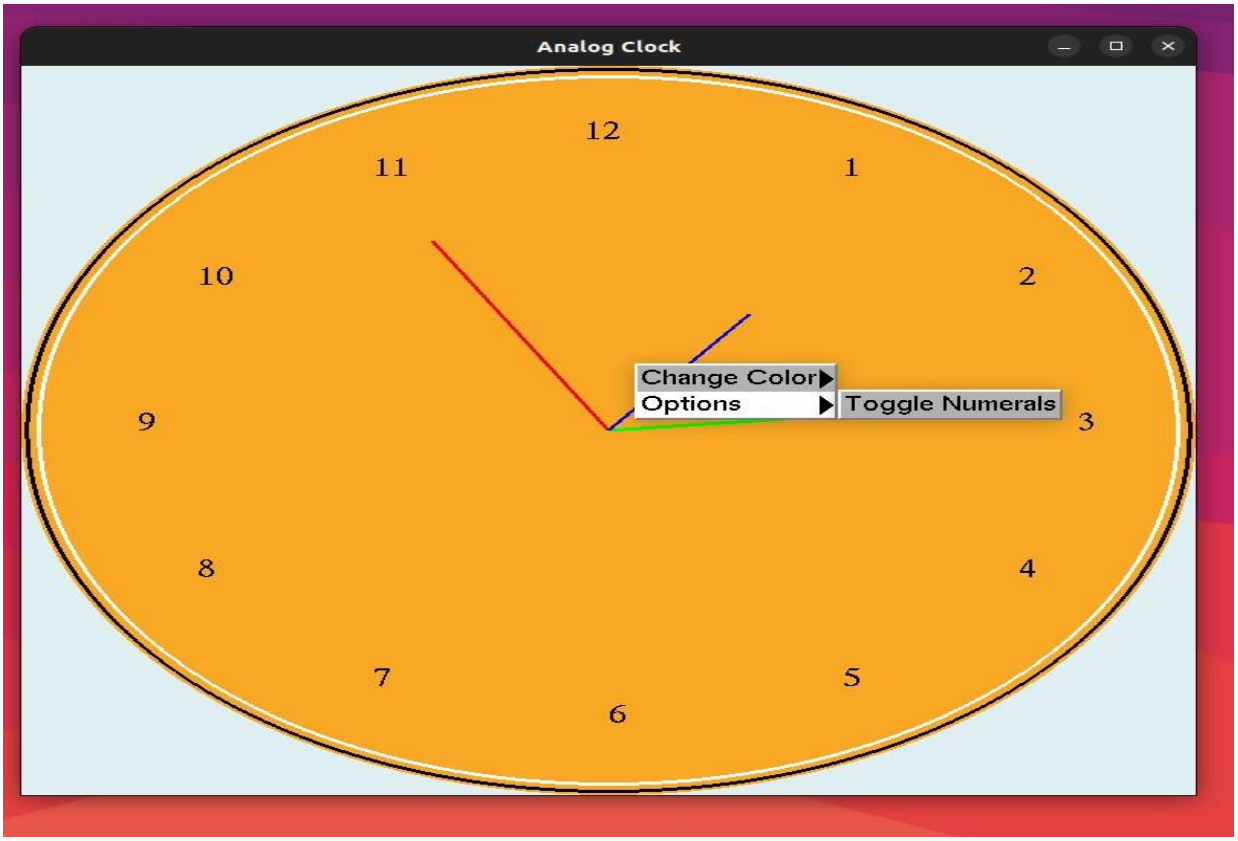

**1. Opening the terminal for code execution**



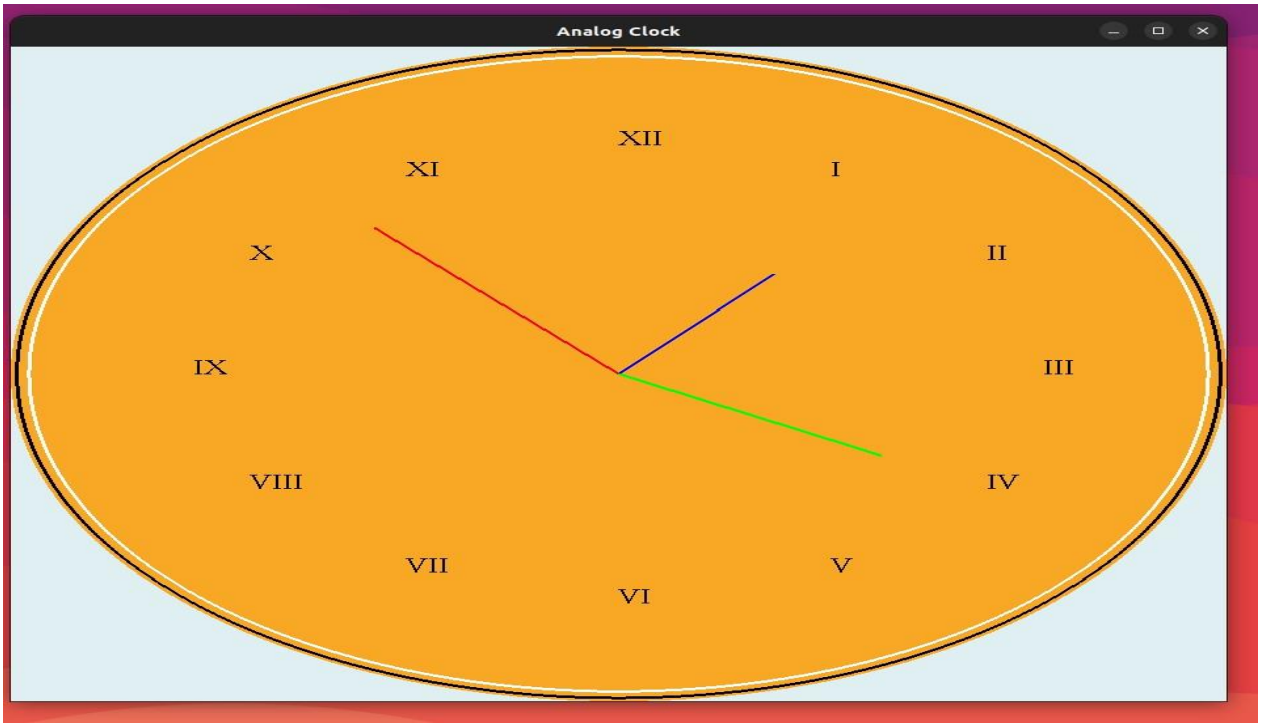

**2.Welcome page**


**3. Starting Screen of Clock**

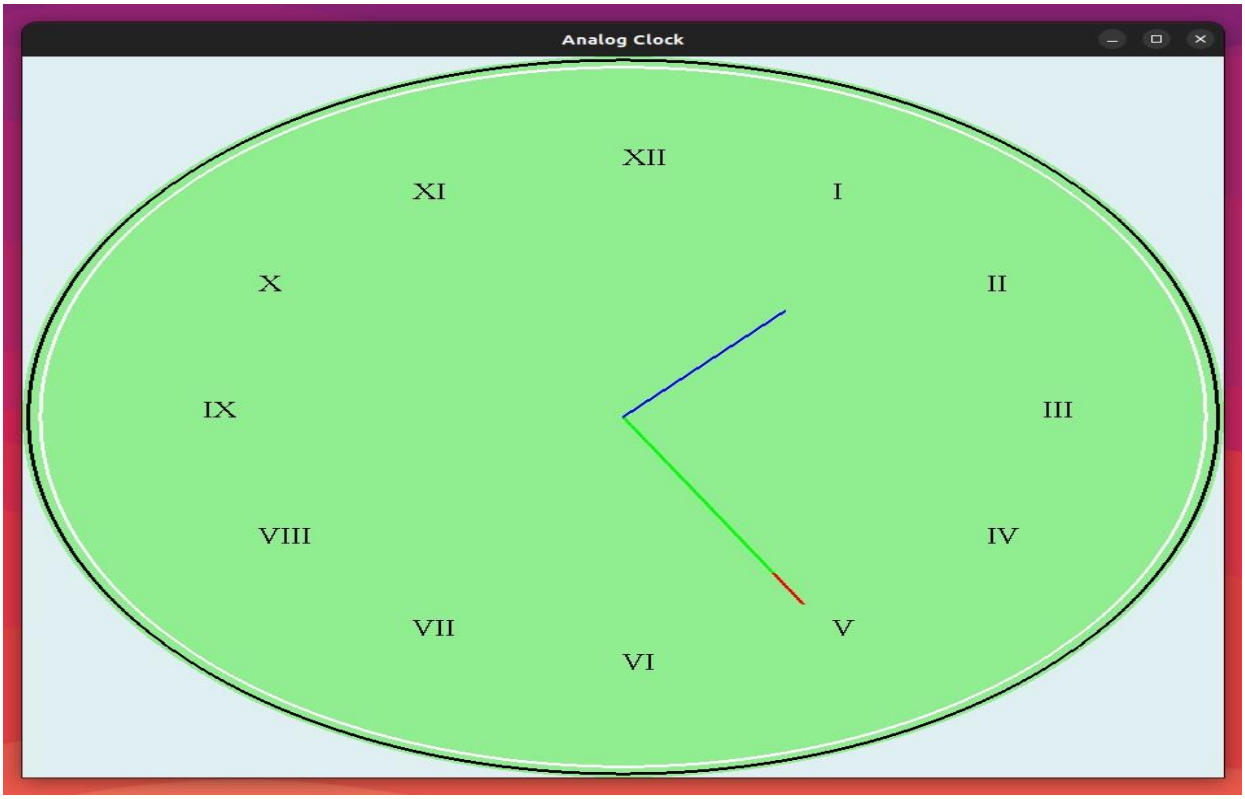

**4. Menu for user to change color of clock**



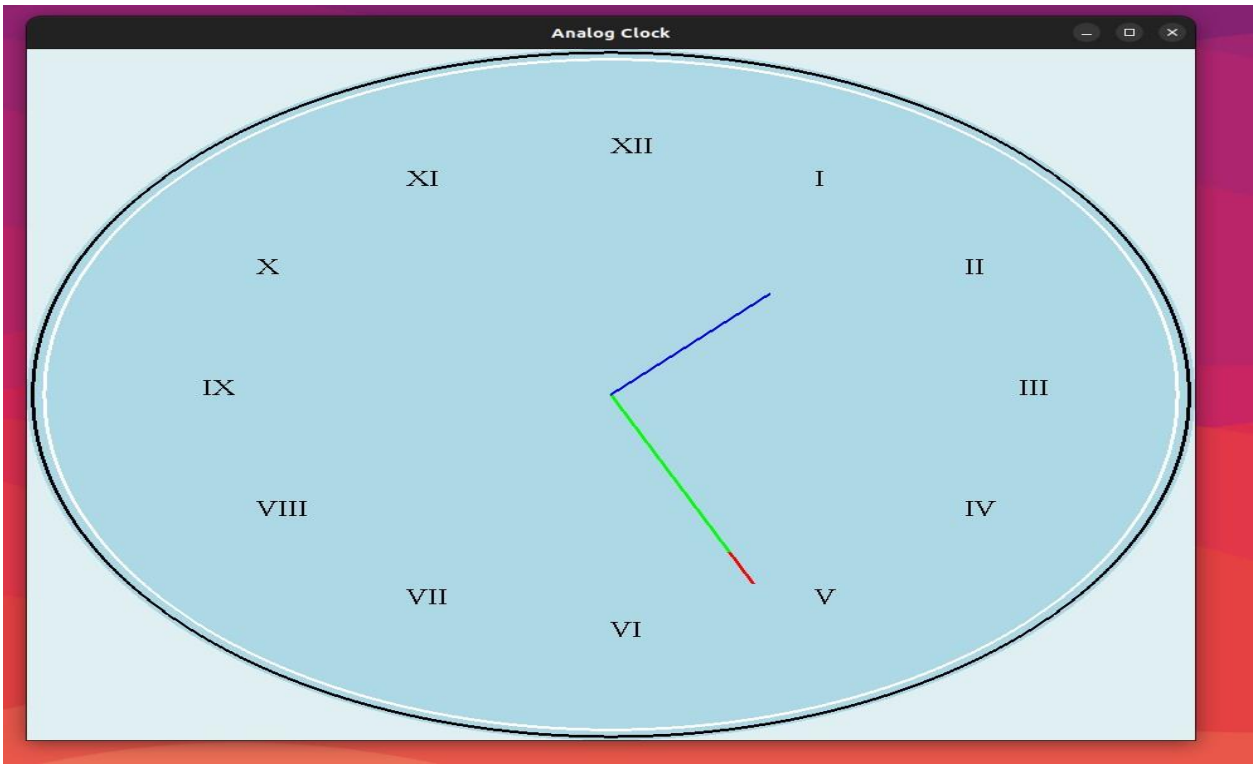

**5.Menu for user to toggle Numeral**




**5. Clock after being toggled in Roman Numeral**




**7. Changing color to Green**




**8.Changing color to Blue**

# Conclusion:

Attempts to represent time in a digital format have been around for ages, with the first patented design of the ‘Plato’ digital clock of 1903 being introduced at the St.
Louis World Fair in 1904. But analogue soldiers on regardless.

Digital displays have enjoyed periods of popularity but people always seem to move back eventually to analogue dials in the long term.

Personally, I place the eternal popularity to be based on the instant recognisability of the position of the hands. with the briefest of glances it is possible to observe the time and a number of other aspects that digits alone do not instantly convey, I always seemed to have to analyse them to get the picture - a dial is of course exactly that - the picture.

The world is an analogous place, we live in an environment of constantly passing time and we unconsciously digest that information as a series of ‘nearly’ ‘just past’ and ‘almost’ approximations of time. That doesn't really work in digits. One of the first things I noticed when experimenting with a digital display was the seconds

It suddenly became far more important. A number is only as good as the ones that surrounded that surrounded it.

So strangely I was always running late. Until I realised that with an analogue display the hands were always moving - as opposed to stepping at intervals. and the brief glance told you everything you needed to know in an instant simply by absorbing the hands positions whereas the digital information is only of elapsed time.

I concluded that as we live an analogous life, reading an analogous dial is synonymous and natural in reading time as it passes and not in steps, it is so much more easily understood in the brief glance, so I have stayed with analogue ever since.

## Future Enhancement:

Here are some future enhancement ideas for your analog clock code:

Customizable Clock Face: Allow users to customize the appearance of the clock face by providing options to change the color, style, and texture of the clock face background.

Multiple Time Zones: Implement the ability to display multiple time zones simultaneously on the clock face. This can be achieved by adding additional clock hands or markings for each time zone.

Alarm Functionality: Add alarm functionality to the analog clock, allowing users to set alarms for specific times. When the alarm time is reached, the clock can provide visual and audible alerts.

Animated Effects: Introduce animated effects to make the clock more visually appealing. For example, you can add smooth transitions when the clock hands move or incorporate subtle animations to the clock face.

Date Display: Include a feature to display the current date alongside the time. This can be done by adding a small digital display or incorporating the date into the clock face design.

User Interaction: Implement interactive features that allow users to interact with the clock. For instance, users can click on specific areas of the clock face to access additional information or perform certain actions.

Theme Support: Introduce theme support to change the overall appearance of the clock. Users can select different themes such as vintage, modern, futuristic, etc., which will modify the design elements of the clock.

Time Format Options: Provide flexibility in displaying time formats. Allow users to choose between a 12-hour or 24-hour format based on their preference.

Background Music: Add an option to play soothing background music while the clock is being displayed, creating a relaxing ambiance.

Integration with External Devices: Explore the possibility of integrating the analog clock with external devices or smart home systems. This could include synchronization with atomic clocks for precise timekeeping or integration with voice assistants for voicecontrolled commands.

# References:

REFERENCE BOOKS:

**[1]** Edward angel: Interactive computer graphics A TOP-DOWN Approach with OpenGL, 2nd edition, Addison-Wesley, 2000.

**[2]** F.S. Hill, Jr.: computer graphics using OpenGL, 2nd edition, Pearson education, 2001**.**

## WEB URL'S:

Http://msdn.microsoft.com http://codeproject.com

http://stackoverflow.com

**********