

**KYUNGDONG UNIVERSITY (KDU)**

**SOUTH KOREA**

**Analog and Digital CLOCK**

**Computer Graphics OPENGL Project**

**Bachelor of Smart computing (4th semester)**

**Prepared By: Submitted To:**

**Dahit Santosh (1724044) Prof. Nur Alam**

**Submission On:**

**2019/06/04**

**ABSTRACT**

This mini project on **CLOCK** displays an analog as well as a digital clock by reading the time from the system in which it runs. It implements the wall clock with the round circular board and three conic geometrics which forms the different (sec, min & hour) hands. Small cubes are also made for the minutes, while each hour is represented with a large block. Digit display of the clock timing is also placed at the bottom of the graphics window.

The ‘localtime (*&ltime)'*function is used to get the local time. With proper mathematics and conversion of the time for sec, min and hours the graphical hands are made to make movement. The user can toggle between two types of views and can enable or disable the background brightness of the clock.

**Table of CONTENTS**

**Contents page no.**

**1. Introduction (01-04)**

1.1. Computer Graphics 01

1.2. Application of Computer Graphics 01

1.3. Introduction to OpenGL 02

1.4. Introduction to GLUT 03

1.5 Applications of OpenGL 04

1.6 OpenGL primitives 04 **2.** **System Requirements (06)**

2.1. Software requirements 06

2.2. Hardware requirements 06

**3. About the Project (06-07)**

3.1. Introduction to the project 0 6

3.2. User defined functions 07

**4. Design (8)**

**5. Implementation (9-21)**

5.1. Various functions used in this program. 9

5.2. Source Code 10

**6. Testing (22)**

* 1. Test Case for Mouse 22
  2. Test Case for Menu Options 22

**7. Snapshots (23-26)**

**8. Conclusion (26)**

**Bibliography (26)**

**LIST OF FIGURES**

**NAME PAGE NO.**

1.3 OpenGL pipeline 02

1.4 Library Organization of OpenGL 03

1.6 OpenGL primitives 04

4.1 Project Design 08

7.1 Snapshot of the initial view of clock 23

7.2 Snapshot of the menu with options for the clock 23

7.3 Snapshot of the ortho view of clock 24

7.4 Snapshot of the perspective view of clock 24

7.5 Snapshot of the clock with brightness on 25

7.6 Snapshot of the clock with brightness off 25

7.7 Snapshot of the window with description 26

**`1. INTRODUCTION**

**1.1 Introduction to Computer Graphics**

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. We can create images that are indistinguishable from photographs of real objects. In other terms, Computer Graphics are the graphics created by the computers, and more generally, the representation and manipulation of image data by a computer.

The development of computer graphics has been driven both by the needs of the user community and by advances in hardware and software.

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 images so produced, and manipulating visual content.

**1.2 Applications of Computer Graphics**

The development of computer graphics has been driven both by the needs of the user community and by advances in hardware and software. The applications of computer graphics are many and varied. We can however divide them into four major areas.

* Display of information: More than 4000 years ago, the Babylonians developed floor plans of buildings on stones. Today, the same type of information is generated by architects using computers. Over the past 150 years, workers in the field of statistics have explored techniques for generating plots. Now, we have computer plotting packages. Supercomputers now allow researchers in many areas to solve previously intractable problems. Thus, Computer Graphics has innumerable applications.
* Design: Professions such as engineering and architecture are concerned with design. Today, the use of interactive graphical tools in CAD, in VLSI circuits, characters for animation have developed in a great way.
* Simulation and animation: One of the most important uses has been in pilots’ training. Graphical flight simulators have proved to increase safety and reduce expenses. Simulators can be used for designing robots, plan it’s path, etc. Video games and animated movies can now be made with low expenses.
* User interfaces: Our interaction with computers has become dominated by a visual paradigm. The users’ access to internet is through graphical network browsers. Thus Computer Graphics plays a major role in all fields.

**1.3 Introduction to OpenGL**

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

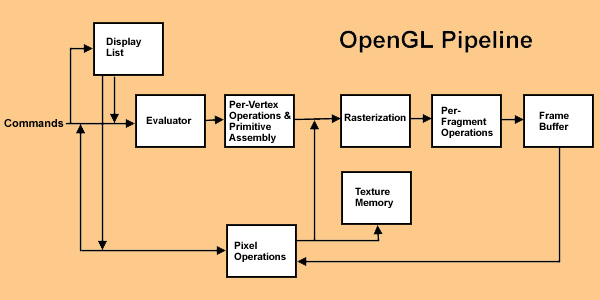
These are certain characteristics of OpenGL:

* OpenGL is a better documented API.
* OpenGL is much easier to learn and program.
* OpenGL has the best demonstrated 3D performance for any API.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it's possible for the API to be implemented entirely in software, it's designed to be implemented mostly or entirely in hardware.

In addition to being language-independent, OpenGL is also platform-independent. The specification says nothing on the subject of obtaining, and managing, an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

OpenGL is an evolving API. New versions of the OpenGL specification are regularly released by the Khronos Group, each of which extends the API to support various new features.In addition to the features required by the core API, GPU vendors may provide additional functionality in the form of *extensions*. Extensions may introduce new functions and new constants, and may relax or remove restrictions on existing OpenGL functions. Vendors can use extensions to expose custom APIs without needing support from other vendors or the Khronos Group as a whole, which greatly increases the flexibility of OpenGL. All extensions are collected in, and defined by, the OpenGL Registry.



1.3 OpenGL Pipeline

**1.4 Introduction to GLUT**

GLUT is the OpenGL utility toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing API for OpenGL. GLUT makes it easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms. GLUT is designed for constructing small to medium sized OpenGL programs.

While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. The GLUT library has both C, C++ (same as C), FORTRAN, and ADA programming bindings. The GLUT source code distribution is portable to nearly all OpenGL implementations and platforms.

GL

GLUT

GLX

Xlib, Xtk

Frame

buffer

OpenGL

Application

program

GLU

1.4 library organization of OpenGL

**1.5 Applications of OpenGL**

* **OpenGL** (**Open G**raphics **L**ibrary)is a cross-language, multi-platform API for rendering 2D and 3D computer graphics.
* The API is typically used to interact with a GPU, to achieve hardware-accelerated rendering.
* It is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games.

**1.6 OpenGL primitives**

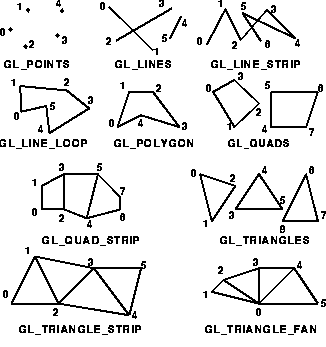
OpenGL supports two classes of primitives:

* Geometric Primitives
* Image(Raster) Primitives

Geometric primitives are specified in the problem domain and include points, line segments, polygons, curves and surfaces.

Raster primitives, such as arrays of pixels pass through a separate parallel pipeline on their way to the frame buffer.

There are ten basic OpenGL primitives:

****

1.6 OpenGL Primitives

**2. HARDWARE AND SOFTWARE REQUIREMENTS**

**2.1 Hardware Requirements:**

* Pentium 90MHz Processor or Higher
* VGA 640x480 or higher-resolution screen supported by Microsoft Windows.
* Recommended 128 MB RAM or Higher (24 MB RAM for Windows 95/98, 32 MB for Windows NT)
* 100GB SATA (Serial Advanced Technology Attachment) Hard Drive
* 5400 RPM hard disk drive
* DirectX 9 capable video card running at 1024 x 768
* DVD-ROM Drive
* Input devices: Keyboard, Mouse.
* Output devices: Monitor.

**2.2** **Software Requirements:**

* Operating System: Microsoft Windows NT 4.0 or later, or Microsoft Windows 95 or later.
* Microsoft Visual Studio 6.0
* glut.h header file
* glut.dll library files

**3. ABOUT THE PROJECT**

**3.1 Introduction to the project**

This mini project on an analog and digital clock displays a real time or local time function that is running on the same device. This project presenting a software based clock on both mode analog and digital. Analog clock consist of a circle shape having number to get time and having hour, minute and second gear to run and indicate the time. It compute normal mathematical calculations and also graphical functions which generate graphs and shows the time in analog as well as digital also.

**3.2 User Defined Functions**

There are eight user defined functions in the source code of an analog and digital clock.

1. **void set\_properties()**

Used to set properties of the surface material, light source properties and the camera position.

1. **Void mouse(int button, int state, int x, int y)**

Displays and updates the needles of the clock as per the current time read by the program.

1. **void Draw\_gear( void )**

Updates the position of the clock gear as per the current matrix stack.

1. **void Draw\_clock( GLfloat cx, GLfloat cy, GLfloat cz )**

Used to draw the analog wall clock on the screen

1. **void num()**

Displays the numbers on the clock according to orthogonal view.

1. **void about()**

Displays a small description about the project when the user clicks the respective mouse button.

1. **void display\_clock()**

Displays the wall clock on the output window.

1. **void options(int id)**

Used to display a menu along with options regarding the light state, view of the clock, description and perform the corresponding action.

**4. DESIGN**

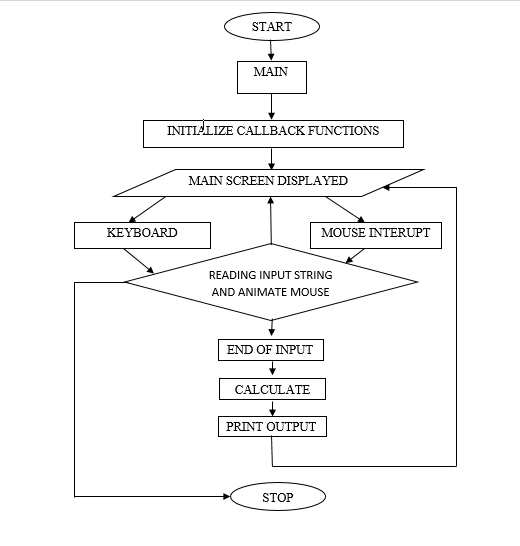
**Initialization**

Initialize the interaction with the windows. Initialize the display mode- double buffer and depth buffer. Initialize the various callback functions for drawing and redrawing, for mouse and keyboard interfaces. Initialize the input and calculate functions for various mathematical calculations. Initialize the window position and size and create the window to display the output.

**Flow of control**

The flow of control in the below flow chart is respected to the Texture Package. For any of the program flow chart is compulsory to understand the program. We consider the flow chart for the texture project in which the flow starts from start and proceeds to the main function after which it comes to the initialization of call back functions and further it proceeds to mouse and keyboard functions, input and calculation functions. Finally, it comes to quit, the end of flow chart.

**4. 1 Project Design**



**5. Implementation**

This program is implemented using various openGL functions which are shown below.

**5.1** **Various functions used in this program.**

* glutInit() : interaction between the windowing system and OPENGL is initiated
* glutInitDisplayMode() : used when double buffering is required and depth information is required
* glutCreateWindow() : this opens the OPENGL window and displays the title at top of the window
* glutInitWindowSize() : specifies the size of the window
* glutInitWindowPosition() : specifies the position of the window in screen co-ordinates
* glutKeyboardFunc() : handles normal ascii symbols
* glutSpecialFunc() : handles special keyboard keys
* glutReshapeFunc() : sets up the callback function for reshaping the window
* glutIdleFunc() : this handles the processing of the background
* glutDisplayFunc() : this handles redrawing of the window
* glutMainLoop() : this starts the main loop, it never returns
* glViewport() : used to set up the viewport
* glVertex3fv() : used to set up the points or vertices in three dimensions
* glColor3fv() : used to render color to faces
* glFlush() : used to flush the pipeline
* glutPostRedisplay() : used to trigger an automatic redrawal of the object
* glMatrixMode() : used to set up the required mode of the matrix
* glLoadIdentity() : used to load or initialize to the identity matrix
* glTranslatef() : used to translate or move the rotation centre from one point to another in three dimensions
* glRotatef() : used to rotate an object through a specified rotation angle
* #define M\_TWOPI 3.1415926535897932384626433832795: for pie value

**5.2 Source Code**

#include <GL/glut.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define M\_TWOPI 3.1415926535897932384626433832795

// define glu objects

int about\_int = 0;

GLUquadricObj \*Cylinder;

GLUquadricObj \*Disk;

struct tm \*newtime;

time\_t ltime;

GLfloat rx, ry, rz, angle;

// lighting

GLfloat LightAmbient[] = { 0.5f, 0.5f, 0.5f, 1.0f };

GLfloat LightDiffuse[] = { 0.5f, 0.5f, 0.5f, 1.0f };

GLfloat LightPosition[] = { 5.0f, 25.0f, 15.0f, 1.0f };

GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

static int light\_state = 1; // light on = 1, light off = 0

static int view\_state = 1; // Ortho view = 1, Perspective = 0

void Sprint(float x, float y, char const \*st)

{ int l, i;

l = strlen(st);

glRasterPos3f(x, y, -1);

for (i = 0; i < l; i++)

{

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, st[i]);

}

static void TimeEvent(int te)

{ rx = 30 \* cos(angle);

ry = 30 \* sin(angle);

rz = 30 \* cos(angle);

angle += 0.01;

if (angle > M\_TWOPI) angle = 0;

glutPostRedisplay();

glutTimerFunc(100, TimeEvent, 1);

}

void init(void)

{ glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_DEPTH\_TEST);

// Lighting is added to scene

glLightfv(GL\_LIGHT1, GL\_AMBIENT, LightAmbient);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, LightDiffuse);

glLightfv(GL\_LIGHT1, GL\_POSITION, LightPosition);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT1);

Cylinder = gluNewQuadric();

gluQuadricDrawStyle(Cylinder, GLU\_FILL);

gluQuadricNormals(Cylinder, GLU\_SMOOTH);

gluQuadricOrientation(Cylinder, GLU\_OUTSIDE);

gluQuadricTexture(Cylinder, GL\_TRUE);

Disk = gluNewQuadric();

gluQuadricDrawStyle(Disk, GLU\_FILL);

gluQuadricNormals(Disk, GLU\_SMOOTH);

gluQuadricOrientation(Disk, GLU\_OUTSIDE);

gluQuadricTexture(Disk, GL\_TRUE);

}

void Draw\_gear(void)

{ int i;

glPushMatrix();

gluCylinder(Cylinder, 2.5, 2.5, 1, 16, 16);

gluDisk(Disk, 0, 2.5, 32, 16);

glTranslatef(0, 0, 1);

gluDisk(Disk, 0, 2.5, 32, 16);

glPopMatrix();

for (i = 0; i < 8; i++)

{ glPushMatrix();

glTranslatef(0.0, 0.0, 0.50);

glRotatef((360 / 8) \* i, 0.0, 0.0, 1.0);

glTranslatef(3.0, 0.0, 0.0);

glutSolidCube(1.0);

glPopMatrix();

}

}

void Draw\_clock(GLfloat cx, GLfloat cy, GLfloat cz)

{ int hour\_ticks, sec\_ticks;

glPushMatrix();

glTranslatef(cx, cy, cz);

glRotatef(180, 1.0, 0.0, 0.0);

/\*glPushMatrix(); // Draw large wire cube (outside of disk clock)

glColor3f(1.0, 1.0, 1.0);

glTranslatef( 0.0, 0.0, 6.0);

glutWireCube(14.0);

glPopMatrix();\*/

glPushMatrix(); // Draw clock face

glTranslatef(0, 0, 1.0);

gluDisk(Disk, 0, 6.75, 32, 16);

glPopMatrix();

glPushMatrix();// Draw hour hand

glColor3f(1.0, 0.5, 0.5);

glTranslatef(0, 0, 0.0);

glRotatef ((360 / 12) \* newtime->tm\_hour + (360 / 60) \* ( 60 / (newtime->tm\_min + 1)), 0.0,0.0,1.0);

glPushMatrix();

glTranslatef(0.0, 0.0, 2.0);

Draw\_gear();

glPopMatrix();

glRotatef(90, 1.0, 0.0, 0.0);

gluCylinder(Cylinder, 0.75, 0, 4, 16, 16);

glPopMatrix();

glPushMatrix();// Draw minute hand

glColor3f(1.0, 0.5, 1.0);

glTranslatef(0, 0, 0.0);

glRotatef((360 / 60) \* newtime->tm\_min, 0.0, 0.0, 1.0);

glPushMatrix();

glTranslatef(0.0, 0.0, 3.0);

glScalef(0.5, 0.5, 1.0);

Draw\_gear();

glPopMatrix();

glRotatef(90, 1.0, 0.0, 0.0);

gluCylinder(Cylinder, 0.5, 0, 6, 16, 16);

glPopMatrix();

glPushMatrix();// Draw second hand

glColor3f(1.0, 0.0, 0.5);

glTranslatef(0, 0, -0.0);

glRotatef((360 / 60) \* newtime->tm\_sec, 0.0, 0.0, 1.0);

glPushMatrix();

glTranslatef(0.0, 0.0, 4.0);

glScalef(0.25, 0.25, 1.0);

Draw\_gear();

glPopMatrix();

glRotatef(90, 1.0, 0.0, 0.0);

gluCylinder(Cylinder, 0.25, 0, 6, 16, 16);

glPopMatrix();

for (hour\_ticks = 0; hour\_ticks < 12; hour\_ticks++)

{ glPushMatrix();// Draw next arm axis.

glColor3f(0.0, 1.0, 1.0); // give it a color

glTranslatef(0.0, 0.0, 0.0);

glRotatef((360 / 12) \* hour\_ticks, 0.0, 0.0, 1.0);

glTranslatef(6.0, 0.0, 0.0);

glutSolidCube(1.0);

glPopMatrix();

}

for (sec\_ticks = 0; sec\_ticks < 60; sec\_ticks++)

{ glPushMatrix();

glTranslatef(0.0, 0.0, 0.0);

glRotatef((360 / 60) \* sec\_ticks, 0.0, 0.0, 1.0);

glTranslatef(6.0, 0.0, 0.0);

glutSolidCube(0.25);

glPopMatrix();

} glPopMatrix();

}

void num()

{

if (view\_state == 1)

{

glColor3f(0.0, 0.0, 1.0);

Sprint(-6.2, -0.2, "9"); //counting from center

Sprint(-0.2, -6.2, "6");

Sprint(-0.4, 5.7, "12");

Sprint(5.8, -0.2, "3");

}

}

void about()

{

//glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 1.0);

Sprint(-5, -2, "This project implements the clock");

Sprint(-5, -2.8, " Both Wall clock and digit clock");

Sprint(-5, -3.6, " is displayed");

Sprint(-5, -4.4, " Clock shows the local time");

Sprint(-5, -5.2, " fetching from computer");

//glFlush();

}

void display\_clock()

{

time(&ltime); // Get time

newtime = localtime(&ltime); // Convert to local time

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// easy way to put text on the screen.

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-8.0, 8.0, -8.0, 8.0, 1.0, 60.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glDisable(GL\_LIGHTING);

glDisable(GL\_COLOR\_MATERIAL);

// Put view state on screen

glColor3f(1.0, 1.0, 1.0);

if (view\_state == 0 && about\_int == 0)

{

Sprint(-3, -4, "Perspective view");

}

else if (view\_state != 0 && about\_int == 0)

{

Sprint(-2, -4, "Ortho view");

}

else

{

about();

}

Sprint(-4, -7.7, asctime(newtime));

// Turn Perspective mode on/off

if (view\_state == 0)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60.0, 1, 1.0, 60.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(rx, 0.0, rz, 0.0, 0.0, -14.0, 0, 1, 0);

}

if (light\_state == 1)

{

glEnable(GL\_LIGHTING);

glEnable(GL\_COLOR\_MATERIAL); // Enable for lighing

}

else

{

glDisable(GL\_LIGHTING);

glDisable(GL\_COLOR\_MATERIAL); // Disable for no lighing

}

Draw\_clock(0.0, 0.0, -14.0);

num();

glutSwapBuffers();

} void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

display\_clock();

glFlush();

}

void reshape(int w, int h)

{ glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

}

void options(int id)

{ switch (id)

{ case 1:

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

about\_int = abs(about\_int - 1);

break;

case 2:

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

view\_state = abs(view\_state - 1);

break;

case 3:

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

light\_state = abs(light\_state - 1);

break;

case 4:

exit(0);

}

}

int main(int argc, char\*\* argv)

{ glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(50, 50);

glutCreateWindow(argv[0]);

glutSetWindowTitle("GLclock");

init();

glutCreateMenu(options);

glutAddMenuEntry("About the Project", 1);

glutAddMenuEntry("Toggle Ortho/Perspective view", 2);

glutAddMenuEntry("Light on/off", 3);

glutAddMenuEntry("Quit", 4);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutTimerFunc(10, TimeEvent, 1);

glutMainLoop();

return 0;

}

**6. TESTING**

Testing in general means validation and verification. It shows that the system conforms to its specifications and system meets all expectation of the user.

6.1 Test case for mouse:

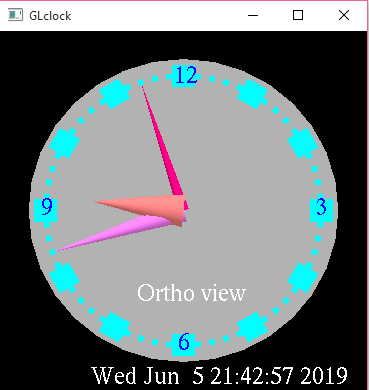
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl. No. | Test Case Description | Expected Result | Actual Result | Remarks |
| 1. | Click the RIGHT mouse button on the display screen. | Menu with   * Toggle Ortho/Perspective View * Light ON/OFF * About the project * Quit   Should be displayed. | Menu with   * Toggle Ortho/Perspective View * Light ON/OFF * About the project * Quit   Is displayed. | Pass |

6.2 Test case for menu options:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl. No. | Test Case Description | Expected Result | Actual Result | Remarks |
| 1. | Click on Toggle Ortho/Perspective view option | The user should be able to toggle between these two views. | The user is able to toggle between these two views. | Pass |
| 2. | Click on Light ON/OFF option | The user should be able to turn ON or OFF the brightness feature. | The user is able to turn ON or OFF the brightness feature. | Pass |
| 3. | Click on About the project option | A short description of the project should be displayed on the window. | A short description of the project is displayed on the window. | Pass |
| 4. | Click on Quit option | The window should be terminated. | The window is terminated. | Pass |

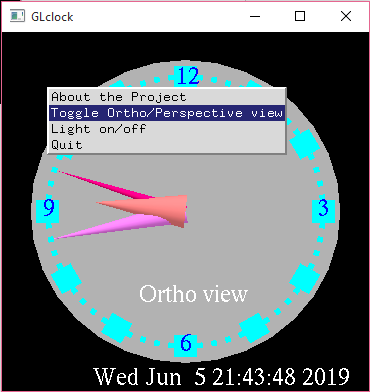
**7. SNAPSHOTS**

**7.1 Initial View:**



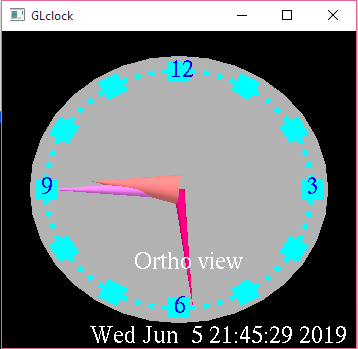
The snapshot above shows the Clock in Ortho view initially.

**7.2 Menu with options:**



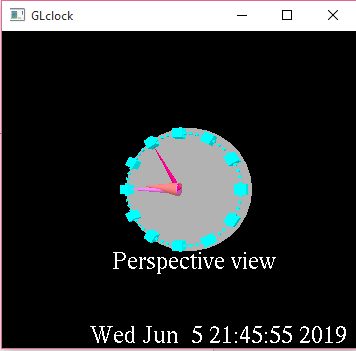
The snapshot above shows the menu containing different options.

**7.3 Ortho view:**



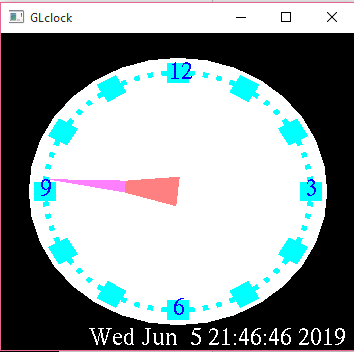
The snapshot above shows the Clock in Ortho view.

**7.4 Perspective view:**

****

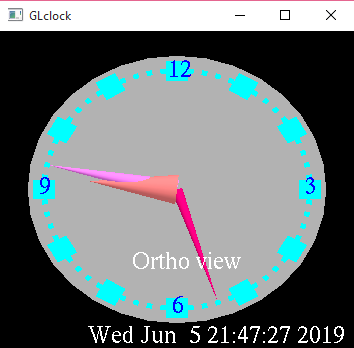
The snapshot above shows the Clock in perspective view. The user is able to toggle between these two views.

**7.5 Brightness ON:**



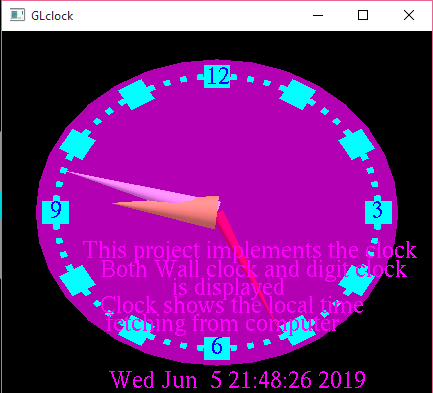
The snapshot above shows the Clock with brightness on.

**7.6 Brightness OFF:**



The snapshot above shows the Clock with brightness off. The user is able to adjust the brightness of the clock.

**7.7 Window with description of the project:**



The snapshot above shows a short description about the project.

1. **CONCLUSION**

This mini project on CLOCK using OpenGL is a reliable graphics package that provides the user with the basic working of an analog as well as a digital clock. It provides the user with certain other operations like toggling between two different kinds of views, adjusting the brightness by turning the lights on or off, also allows the user to read a short description of what the project is about. The user-friendly interface allows the user to interact with it very effectively.

**BIBLIOGRAPHY**

BOOKS:

* Edward Angel, “Interactive Computer Graphics”,5th edition, Pearson Education,2005
* Donald D Hearn and M. Pauline Baker, “Computer Graphics with OpenGL”, 3rd edition.

WEBSITES:

* <http://www.opengl.org> <http://openglprojects.in>
* <http://www.wikipedia.com>
* <http://basic4gl.wikispaces.com>