VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590 018

A MINI PROJECT REPORT

on

STEAM ENGINE

Submitted in partial fulfillment of the requirement for the curriculum of the 6th semester

Bachelor of Engineering
In
Computer Science & Engineering

by

Chirag Radhakrishna (1VI19CS021)
Chiranjeevi PR (1VI19CS022)

Under the supervision of

Mr. Noor Basha
Assistant Professor

Mr. Naveen H S
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
VEMANA INSTITUTE OF TECHNOLOGY
Bengaluru-560034
2021-2022

Karnataka Reddyjana Sangha®

VEMANA INSTITUTE OF TECHNOLOGY

Koramangala, Bengaluru-34.

(Affiliated to Visvesvaraya Technological University, Belagavi)

Department of Computer Science & Engineering

Certificate

This is certified that the Computer Graphics mini project work entitled **"Steam Engine"** carried out by **Chirag Radhakrishna (1VI19CS021) and Chiranjeevi PR (1VI19CS022)** are bonafide students of **Vemana Institute of Technology** in partial fulfillment of the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belagavi during the academic year 2021-2022. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the said degree.

__________________	__________________	__________________
Guide-I	**Guide-II**	**HOD**
Mr. Noor Basha	Mr. Naveen H S	Dr. M Ramakrishna

External Viva

Name of the Examiners	Signature with date
1.____________________	1.____________________
2.____________________	2.____________________

ACKNOWLEDGEMENT

We sincerely thank **Visvesvaraya Technological University** for providing a platform to do the mini project.

Firstly, we would like to express our deep sense of gratitude to our institute "**VEMANA INSTITUTE OF TECHNOLOGY**" that provided us an opportunity to do a project entitled **"Steam Engine"**.

We thank **Dr. Vijayasimha Reddy. B.G,** Principal, Vemana Institute of Technology, Bengaluru for providing the necessary support.

We would like to place on record our regards to **Dr. M. Ramakrishna,** Head of the Department Computer Science and Engineering for his continued support.

We would like to thank our project guide **Mr. Noor Basha,** Assistant Professor, Dept. of CSE and **Mr. Naveen H S,** Assistant Professor, Dept. of CSE for their continuous support and valuable guidance towards successful completion of the project.

We would be failing in our duty if we do not thank the faculty members, lab staffs, technicians and family members for their constant support and guidance.

Date: **Chirag Radhakrishna (1VI19CS021)**

Place: BANGALORE **Chiranjeevi PR (1VI19CS022)**

ABSTRACT

Our mini project is used to depict a simulation model of a STEAM ENGINE. The model consists of the various parts that constitute the engine. The engine piston, connecting rods, crankshaft are designed to rotate either in the clockwise or anti-clockwise direction. A shading and texture provision is also available which allows viewing of the engine in different perspectives. Also, the speed of rotation can be increased/decreased. All these operations are mainly performed using keyboard keys. Assembling the parts of the engine together requires extensive usage of transformation functions, mainly translation and rotation. This mini project highlights a few features of the components of a Steam Engine.

CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Computer graphics is concerned with all aspects of producing picture or an image using computers and, more generally, the representation and manipulation of pictorial data by a computer. The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. Such graphs are used to illustrate papers, reports, thesis, and other presentation material. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines. We implement computer graphics using the OpenGL API. OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics.

OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn). Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimensional objects such as a sphere, a torus, and even a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but it is a useful starting point for learning OpenGL.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted too various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT routines also take relatively few parameters.

1.1 Computer Graphics

The term computer graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

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

Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. Fig 1.1 shows how inputs go to the processor, from there to the frame buffer and then it is displayed.

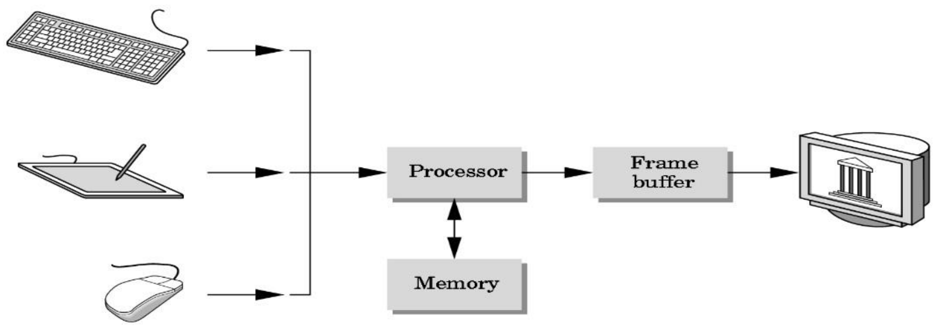

Fig 1.1 A Graphics System

1.2 OpenGL Technology

OpenGL is strictly defined as "a software interface to graphics hardware." In essence, it is a 3D graphics and modelling library that is highly portable and very fast. Using OpenGL, you can create elegant and beautiful 3D graphics with exceptional visual quality. The greatest advantage to using OpenGL is that it is orders of magnitude faster than a ray tracer or software-rendering engine. Initially, it used algorithms carefully developed and optimized by Silicon Graphics, Inc. (SGI), an acknowledged world leader in computer graphics and animation.

Fig 1.2 shows it has four major steps in pipeline architecture:

1. Vertex processing: Much of the work in the pipeline is in converting object representations from one co-ordinate system to another. Every vertex is processed independently.

The co-ordinate system includes the following:

- Object co-ordinates
- Camera co-ordinates
- Screen co-ordinates.

2. Clipping and Primitive assembly: clipping is necessary, because of the limitation that no imagining system can see the whole object at once. Cameras have film of limited size. Image outside the clipping volume is clipped out. Clipping is done by vertex basis.

3. Rasterization: generation of pixels in the frame buffer. Rasterizer determines which pixels in the frame buffer are inside the polygon. The output of the Rasterizer is a set of fragments for each primitive. Fragments carry color, location, depth and other information.

4. Fragment processing: Updates the pixel in the frame buffer. Color of the pixels that correspond to fragment can be read from the frame buffer. Color fragment may be altered by texture mapping.

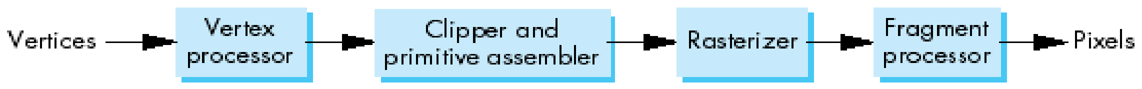

Fig 1.2 Graphics pipeline.

OpenGL Fundamentals

The section explains some of the concepts inherent in OpenGL. Primitives and Commands OpenGL draws primitives-points, line segments, or polygons-subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer).

Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normals, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be clipped so that a particular primitive fits within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order, in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.) Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene. Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, obtained by pasting a texture onto the objects, or some combination of these three actions.

Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called rasterization.

OpenGL is a standard specification that defines an API that is multi-language and multi-platform and that enables the codification of applications that output computerized graphics in 2D and 3D.

The interface consists in more than 250 different functions, which can be used to draw complex tridimensional scenes with simple primitives. It consists of many functions that help to create a real-world object and a particular existence for an object can be given.

1.3 Project Introduction- Steam Engine

The aim of this project is to create a Steam Engine. The Engine is made up of a Piston, Engine Pole, Cylinder Head, Flywheel, Crank Bell and a Crank. The viewer is allowed to rotate the Engine's Crank either in clock wise or in anti-clock wise direction. The viewer can also slow up or slow down the Crank speed. First a Piston, Engine Pole, Cylinder Head, Flywheel, Crank Bell and a Crank is created using myCylinder() function. The main primitives used inside the myCylinder() function to create a Cylinder is gluCylinder( ) and gluDisk( ) . So every time, myCylinder( ) function is called inside the functions used to create Piston, Engine Pole, Cylinder Head, Flywheel, Crank Bell and Crank. The parts mentioned above are combined to form a Steam Engine image. We can make Steam Engine transparent and display. In display function, at first it clears the drawing buffer and if transparency is set, displays the model twice, first time accepting those fragments with a ALPHA value of 1 only, then with DEPTH_BUFFER writing disabled for those with other values. Initially when the animation is not called, the crank angle will not change and the window is idle. When called, it increments the crank angle by ANGLE_STEP, updates the head angle and notifies the system that the screen needs to be updated. When a menu option has been selected, it translates the menu item identifier into a keystroke, then calls the keyboard function. A menu will be associated with the mouse too. The viewer can also see the shaded and textured steam engine.

The controls are: -

1. 'a' - > To rotate crank anti-clock wise.
2. 'z' - > To rotate crank clock wise.
3. '+' and '-' - > To speed up and speed down
4. 'o' - > Transparency.
5. '0' and '1' - > Right light and Left light respectively
6. 's' and 't' - > Shading and Texture respectively

CHAPTER 2

SYSTEM REQUIREMENTS SPECIFICATION

2.1 Minimum hardware requirements

- **Microprocessor:** 1.0 GHz and above CPU based on either AMD or INTEL Microprocessor Architecture.
- **Main memory:** 512 MB RAM.
- **Hard Disk:** 40 GB.
- **Hard disk speed in RPM:** 5400 RPM.
- **Keyboard:** QWERTY Keyboard.
- **Mouse:** 2 or 3 Button mouse.
- **Monitor:** 1024 x 768 display resolution.

2.2 Minimum software requirements

- **Operating system**: Windows XP and later.
- Visual C++ 2005 and later.
- OPENGL Library.
- X86.
- X64(WOW).
- Mouse Driver.
- Graphics Driver.
- C Libraries.

CHAPTER 3

DESIGN

3.1 System design

The mini project is designed to how graphics can be used to represent data. It is a graphics package encoded in Microsoft Visual C++ with OpenGL and the mini project basically deals with providing the graphical representation of data that a user provides as statistics to the system.

The above mini project can be analyzed as follows:

- The mini project is implemented using ‘C’ and ‘OpenGL’ built in functions.
- In the beginning the user is asked to enter the statistical data through the windows command line.
- Once the data is input, the computations are done by program to compute various co-ordinates etc. and appropriate graph as per users choice and the output is rendered on the display window.
- This project is developed using Microsoft Visual C++ with OpenGL as an academic project using keyboard and mouse. This project is implemented by making extensive use of library functions offered by graphic package of ‘OpenGL’.
- A list of standard library functions that are used are given below. A number of user defined functions also have been used and a summary of those functions follows this list of standard library functions.
- The aim of the project is to show how the steam engine components are assembled and how it can be viewed with respect to different perspectives. This has been implemented using computer graphics.
- The various functions used to implement it are draw_engine_pole( ), draw_piston( ), draw_cylinder( ), draw_flywheel( ) and draw_crankbell( ).

3.2 Data Flow Graph

- The myCylinder ( ) function is used to draw a cylinder using glu function, drawing flat disc's at each end, to give the appearance of it being solid.
- The draw_piston( ) function is used to draw a piston.
- The draw_engine_pole( ) function is used to draw the engine pole and the pivot pole for the cylinder head.
- The draw_cylinder_head( ) function draws the cylinder head at the appropriate angle, does the necessary translations for the rotation.
- The draw_flywheel( ) function draws the flywheel.
- The draw_crankbell( ) function is used to draw the crank bell, and the pivot pin for the piston. Also calls the appropriate display list of a piston that performs the necessary rotations beforehand.
- The draw_crank( ) function draws the complete crank. Piston also gets drawn through this function.
- The display( ) function is used for the Main display routine. Clears the drawing buffer and if transparency is set, it displays the model twice, first time, accepting those fragments with an ALPHA value of 1 only, then with DEPTH_BUFFER writing disabled for those with other values.
- The animation( ) function is called when the window is idle. When called, it increments the crank angle by ANGLE_STEP, updates the head angle and notifies the system that the screen needs to be updated.
- The project is implemented on C platform with the help of OpenGL built-in functions.

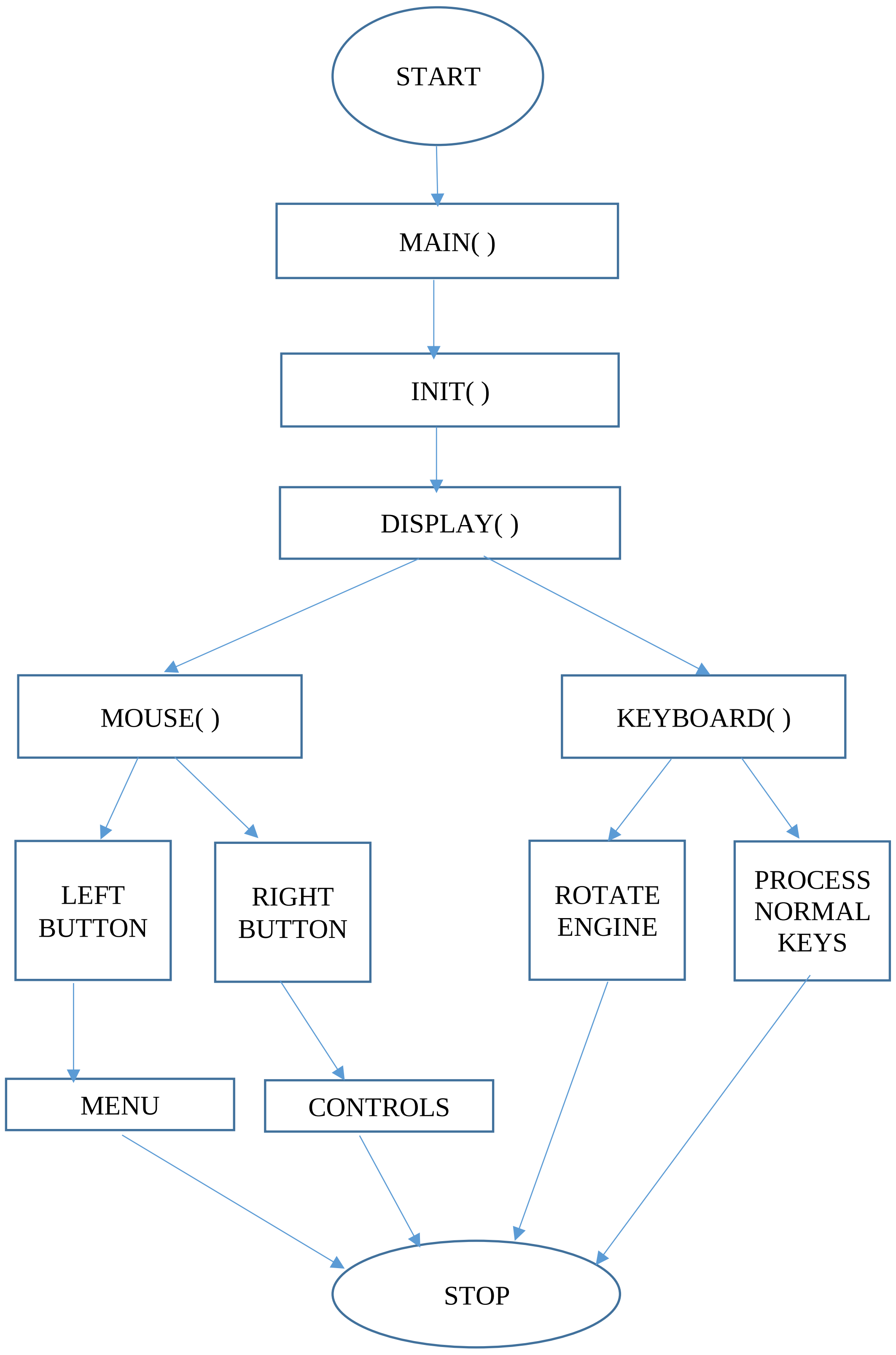

Fig 3.1 Data flow graph

CHAPTER 4

IMPLEMENTATION

4.1 Introduction to visual studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows super family of operating systems, as well as websites, web applications and web services. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level, including adding support for source-control systems (like Subversion) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++ and C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). Support for other languages such as M, Python, and Ruby among others is available via language services installed separately.

4.2 Algorithm

1. Initialize the display mode ;
2. Initialize the window position and size;
3. Create display window;
4. Initialize the keyboard function;

5. Initialize mouse function;
6. Call display function ;
7. Call main loop( );

4.3 OpenGL Functions

The functions used in the programs are as follows:

- **main():** The control will first enter into this function and in this function it will create a window and it will initialize the window and then this function calls display function. Helps implement a miniature model of a steam engine. Sets system in double buffered mode and initializes all the call-back functions.
- **void glScalef(TYPE sx, TYPE sy, TYPE sz)**: alters the current matrix by a scaling of (sx, sy, sz). TYPE here is GLfloat. Here, we use scaling to minimize the length of the curve at each iteration. For this curve we use the scale factor to be 3 units because we substitute a line by 4 lines in each iteration.
- **void glRotatef(TYPE angle, TYPE dx, TYPE dy, TYPE dz)**: alters the current matrix by a rotation of angle degrees about the axis(dx, dy, dz). TYPE here is GLfloat. For a Koch curve we rotate by 60° about the z-axis.
- **void glTranslatef(TYPE x, TYPE y, TYPE z)**: alters the current matrix by a displacement of (x, y, z). TYPE here is GLfloat. We need to translate to display the new position of the line from the old position and also to go out to the beginning of the next side while drawing.
- **void menu(int val)**: Called when a menu option has been selected. Translates the menu item identifier into a keystroke, then call's the keyboard function.
- **void create_menu()**: Initialises the menu of toggles.
- **void make_image(void)**: Makes a simple check pattern image.
- **void make_table(void)**: Makes the head look up table for all possible crank angles.
- **void myinit(void)**: Initialises texturing, lighting, display lists, and everything else associated with the model.
- **void special(int key, int x, int y):** helps the user view the steam engine from different perspectives.

- **void glutMouseFunc(myMouse)**: refers to the mouse callback function. The function to callback is defined as-

  ```
   void myMouse(int button, int state, int x)
  {
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
  if (glutGetModifiers() & GLUT_ACTIVE_SHIFT)
          decrease a level of recursion
  else
          increase a level of recursion
  }
  ```

  Here mouse interface is given to increase a level of recursion by clicking mouse button and also to decrease a level of recursion by doing the same holding the Shift key on the keyboard.
- **void glutKeyboardFunc(myKey)**: refers to the keyboard callback function. The function to callback is defined as-

  ```
  void myKey(unsigned char key, int x, int y)
  {
  if (c == 't')
  exit
  if (c == 's')
  //STATEMENTS and repeat when finished
  }
  ```

  Here keyboard interface is given to quit, the user can quit by pressing ‘q’ and to see next example of the implementation, the user should press ‘n’.

4.4 Module Description

● void myKeyboard(…);

This is the keyboard callback function. Within the callback function, we define the actions that we want to take place if the specified event occurs.

Declaration:

```
void keyboard(unsigned char key, int x, int y)
{
        switch (key) {
        case 's':
                if (shaded == FALSE)
                {
                        shaded = TRUE;
                        glShadeModel(GL_SMOOTH);
                        glEnable(GL_LIGHTING);
                        glEnable(GL_DEPTH_TEST);
                        glEnable(GL_COLOR_MATERIAL);
                        gluQuadricNormals(obj, GLU_SMOOTH);
                        gluQuadricDrawStyle(obj, GLU_FILL);
                }
                else
                {
                        shaded = FALSE;
                        glShadeModel(GL_FLAT);
                        glDisable(GL_LIGHTING);
                        glDisable(GL_DEPTH_TEST);
                        glDisable(GL_COLOR_MATERIAL);
                        gluQuadricNormals(obj, GLU_NONE);
                        gluQuadricDrawStyle(obj, GLU_LINE);
                        gluQuadricTexture(obj, GL_FALSE);
                }
                if (texture && !shaded);
                else
                        break;
        case 't':
                if (texture == FALSE)
                {
```

```
                texture = TRUE;
                glEnable(GL_TEXTURE_2D);
                gluQuadricTexture(obj, GL_TRUE);
        }
        else
        {
                texture = FALSE;
                glDisable(GL_TEXTURE_2D);
                gluQuadricTexture(obj, GL_FALSE);
        }
        break;
case 'o':
        if (transparent == FALSE)
        {
                transparent = TRUE;
        }
        else
        {
                transparent = FALSE;
        }
        break;
case 'a':
        if ((crank_angle += crank_step) >= 360)
                crank_angle = 0;
        head_angle = head_look_up_table[crank_angle];
        break;
case 'z':
        if ((crank_angle -= crank_step) <= 0)
                crank_angle = 360;
        head_angle = head_look_up_table[crank_angle];
        break;
case '0':
        if (light1)
        {
                glDisable(GL_LIGHT0);
                light1 = FALSE;
        }
        else {
                glEnable(GL_LIGHT0);
                light1 = TRUE;
        }
        break;
case '1':
        if (light2)
        {
                glDisable(GL_LIGHT1);
                light2 = FALSE;
        }
        else
        {
                glEnable(GL_LIGHT1);
                light2 = TRUE;
        }
        break;
case '4':
        if ((view_h -= ANGLE_STEP) <= 0)
                view_h = 360;
        break;
case '6':
        if ((view_h += ANGLE_STEP) >= 360)
                view_h = 0;
        break;
case '8':
        if ((view_v += ANGLE_STEP) >= 360)
```

```
                        view_v = 0;
                break;
        case '2':
                if ((view_v -= ANGLE_STEP) <= 0)
                        view_v = 360;
                break;
        case ' ':
                if (anim)
                {
                        glutIdleFunc(0);
                        anim = FALSE;
                }
                else
                {
                        glutIdleFunc(animation);
                        anim = TRUE;
                }
                break;
        case '+':
                if ((++crank_step) > 45)
                        crank_step = 45;
                break;
        case '-':
                if ((--crank_step) <= 0)
                        crank_step = 0;
                break;
        default:
                return;
        }
        glutPostRedisplay();
}
```

- **Display Function**

This function is used to display the main screen of the project.

Declaration:

```
void display(void)
{
        int pass;
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        if (transparent)
        {
                glEnable(GL_ALPHA_TEST);
                pass = 2;
        }
        else
        {
                glDisable(GL_ALPHA_TEST);
                pass = 0;
        }
        glRotatef(view_h, 0, 1, 0);
        glRotatef(view_v, 1, 0, 0);
        do
        {
                if (pass == 2)
                 {
                        glAlphaFunc(GL_EQUAL, 1);
                        glDepthMask(GL_TRUE);
                        pass--;
                  }
                else if (pass != 0)
                {
                        glAlphaFunc(GL_NOTEQUAL, 1);
                        glDepthMask(GL_FALSE);
```

```
                                pass--;
                        }
                        draw_engine_pole();
                        glPushMatrix();
                        glTranslatef(0.5, 1.4, 0.0);
                        draw_cylinder_head();
                        glPopMatrix();
                        glPushMatrix();
                        glTranslatef(0.0, -0.8, 0.0);
                        draw_crank();
                        glPopMatrix();
                } while (pass > 0);
                glDepthMask(GL_TRUE);
                glutSwapBuffers();
                glPopMatrix();
        }
```

CHAPTER 5

SAMPLE OUTPUT

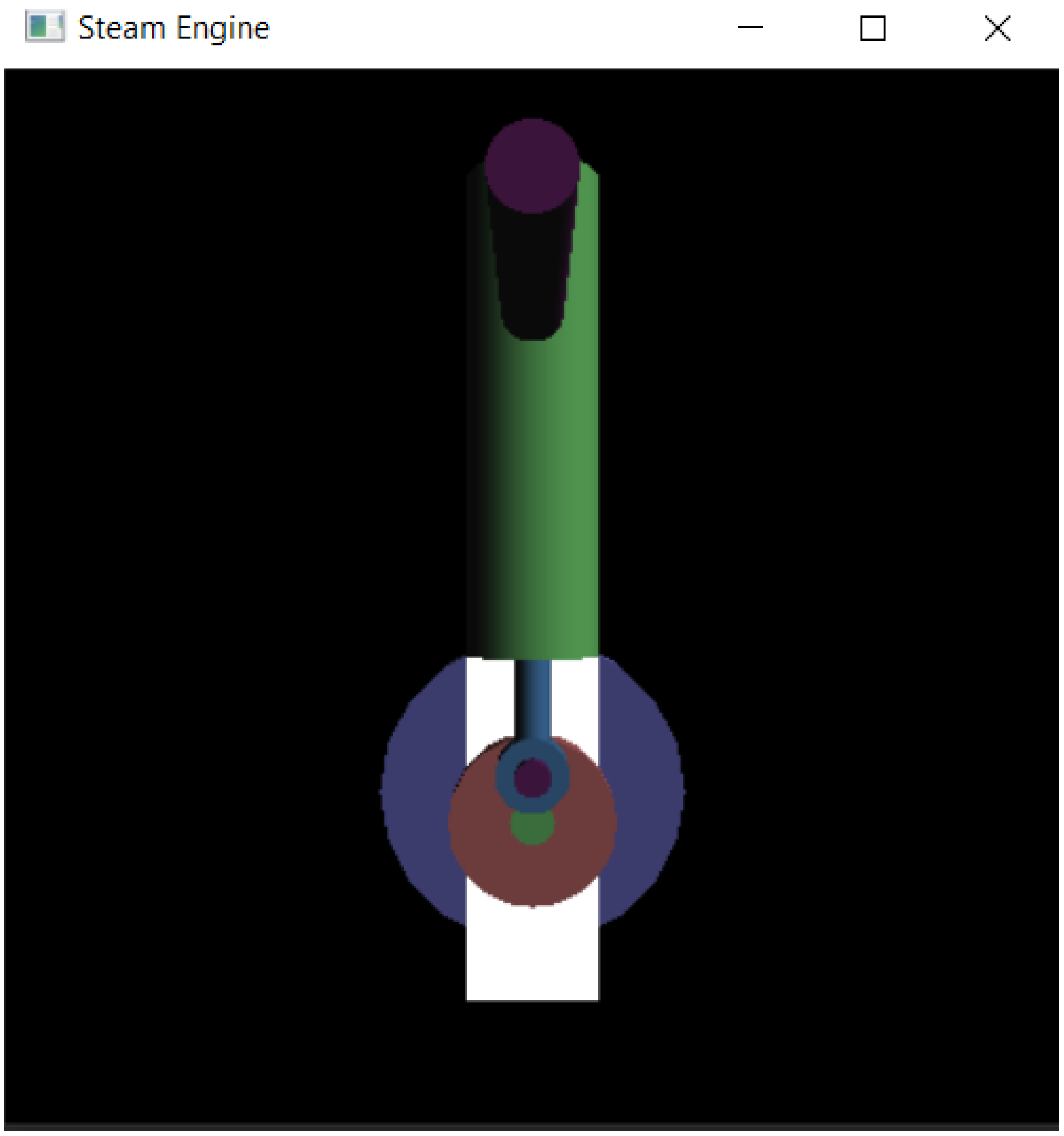

Fig 5.1 Initial View

Fig 5.1 represents the initial view of the steam engine, where the constituent parts have been assembled together.

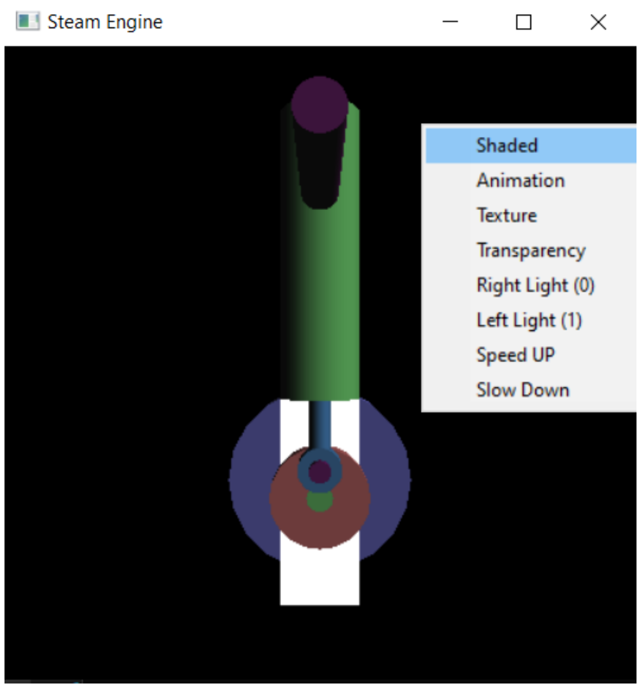

Fig 5.2 Menu associated with mouse

Fig 5.2 represents the various options that can be selected for perspective viewing of the engine.

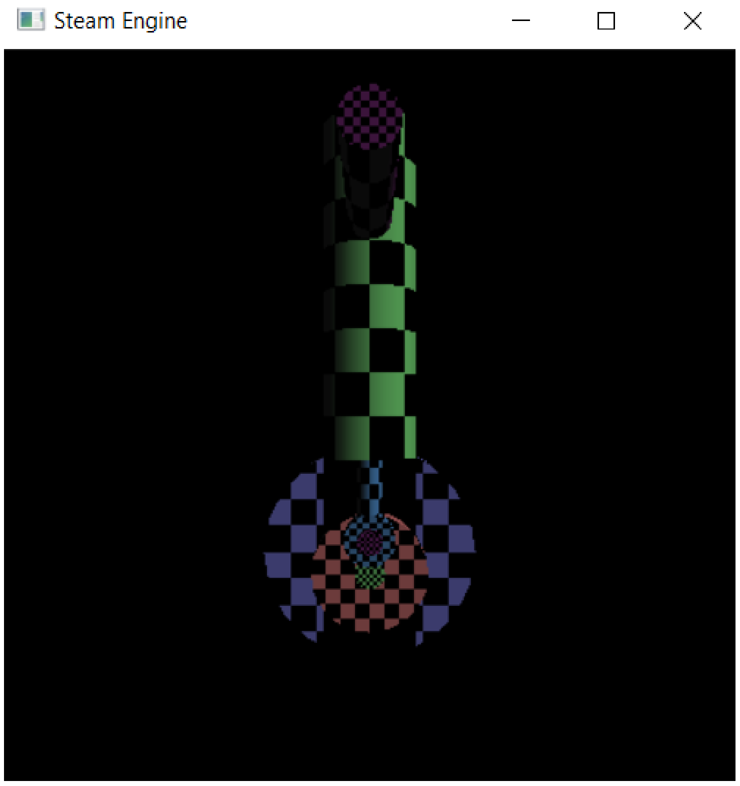

Fig 5.3 Texture

Fig 5.3 shows the resulting engine view when the texture is applied to the different parts.

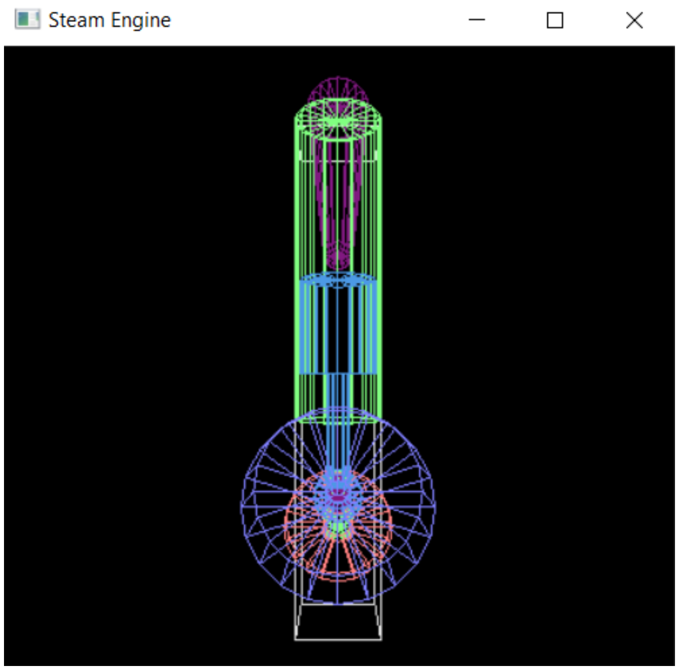

Fig 5.4 Shading

Fig 5.4 shows the resulting engine view when shading is applied to the different parts. We obtain a blueprint-like display of the engine where we see the engine structure.

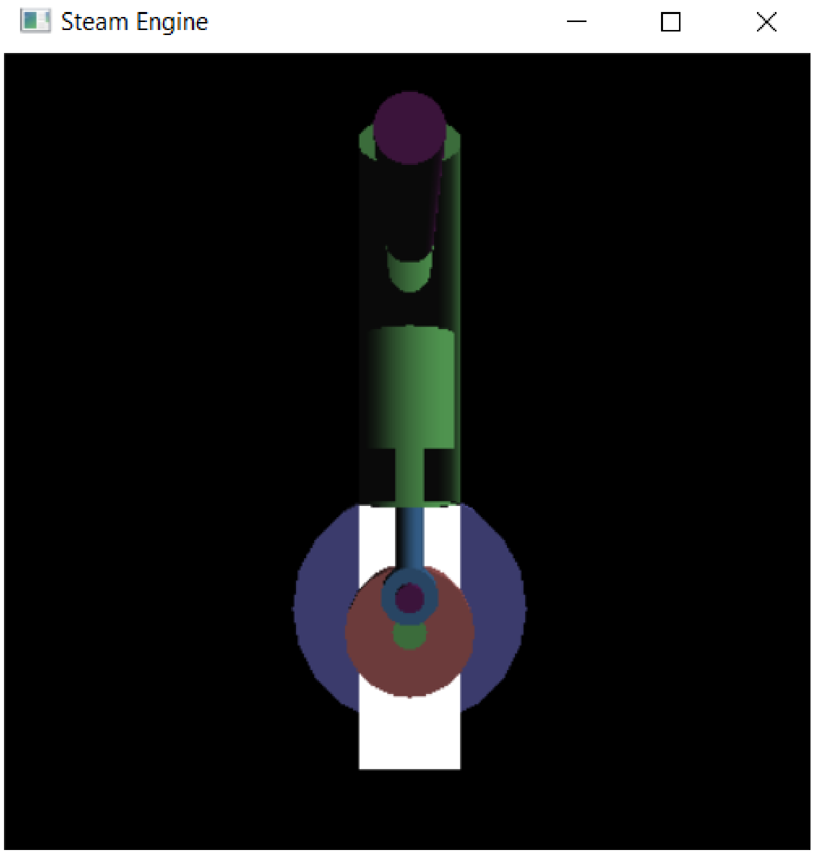

Fig 5.5 Transparency: selecting this option provides a transparent view of the engine.

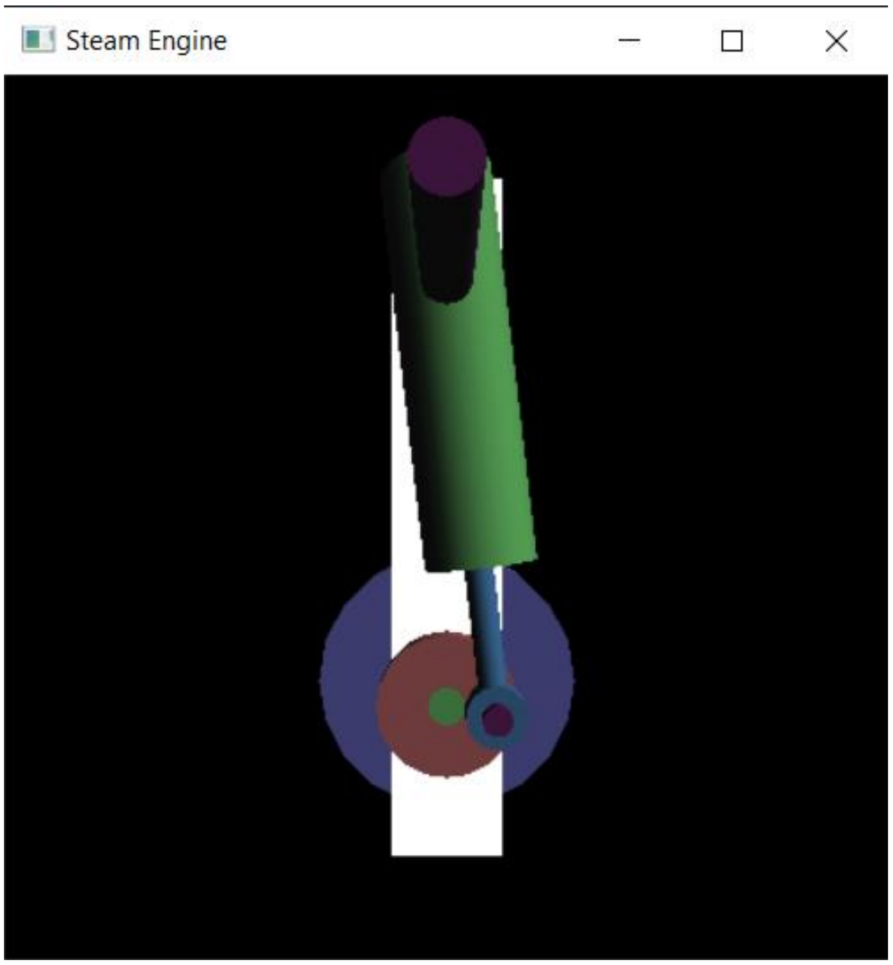

Fig 5.6 Rotation of steam engine

Fig 5.6 depicts the rotation of steam engine. The piston can be rotated in both the clockwise and counter-clockwise direction.

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion

“Steam Engine” has been developed using OpenGL package. Various display functions and OpenGL built-in functions have been used for the development. The different components of the steam engine were developed with some features like translation and rotation and were assembled together. The project takes the help of a keyboard to operate the functions to rotate and increase the speed of the engine and also to apply shading and texture. Components like the piston, crank bell, cylinder head and flywheel were developed using 3D properties. Various transformation functions like animation, scaling and shearing are used for making the objects clearer and more visible based on dimensions. A user-defined menu function has been implemented to enable access to the different options. By developing this “Steam Engine” using OpenGL we have learnt the OpenGL package in Microsoft Visual Studio 2005.

Future Enhancement

The project can be further enhanced by adding some of the features using OpenGL in future. The project can be further implemented by including the background effect and translate command along with the Keyboard interactions been given. Various transformations can be applied to demonstrate the assembly phase of the steam engine where the components are placed accordingly to obtain the engine design. The project can further be implemented by other modification if required.

BIBLIOGRAPHY

Books

- Edward Angel’s Interactive Computer Graphics Pearson Education 5th Edition
- Interactive computer Graphics --A top down approach using open GL--by Edward Angel.
- Jackie.L.Neider,MarkWarhol,Tom.R.Davis,"OpenGL Red Book",Second Revised Edition, 2005.
- Donald D Hearn and M.PaulineBaker,"Computer Graphics with OpenGL", 3rd Edition.

Websites

- www.OpenGL Redbook.com
- www.OpenGL simple examples.
- www.OpenGL programming guide.
- http://www.wikipedia.com
- http://basic4gl.wikispaces.com
- http://www.opengl.org
- http://pyopengl.sourceforge.net/documentation/manual/glut.3GLUT.htm