VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

Mini Project Report on

"STEAM ENGINE"

Submitted in the partial fulfillment for the requirements of Computer Graphics & Visualization Laboratory of 6th semester CSE requirement in the form of the Mini Project work

Submitted By

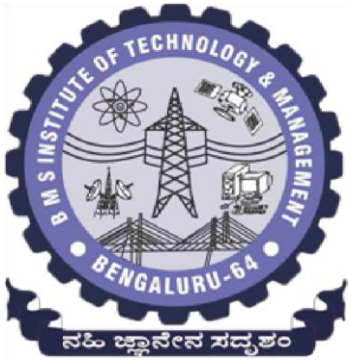
KUSHAL SR USN: 1BY18CS080

PRATEEK USN: 1BY15CS111

SANTOSH PM USN: 1BY18CS144

Under the guidance of

Mr. SHANKAR R
Assistant Professor, CSE, BMSIT&M



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT

YELAHANKA, BENGALURU - 560064.

2019-2020

BMS INSTITUTE OF TECHNOLOGY &MANAGEMENT

YELAHANKA, BENGALURU – 560064

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project work entitled **"STEAM ENGINE"** is a bonafide work carried out by **KUSHAL SR(1BY18CS080), PRATEEK(1BY18CS111) and SANTOSH PM(1BY18CS144)**in partial fulfillment for *Mini Project* during the year 2020-2021. It is hereby certified that this project covers the concepts of *Computer Graphics & Visualization*. It is also certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report.

Signature of the Guide with date
Mr. SHANKAR R
Assistant Professor
CSE, BMSIT&M

Signature of HOD with date
Dr. Anil G N
Prof & Head
CSE, BMSIT&M

INSTITUTE VISION

To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

INSTITUTE MISSION

Accomplish stimulating learning environment through high quality academic instruction, innovation and industry-institute interface.

DEPARTMENT VISION

To develop technical professionals acquainted with recent trends and technologies of computer science to serve as valuable resource for the nation/society.

DEPARTMENT MISSION

Facilitating and exposing the students to various learning opportunities through dedicated academic teaching, guidance and monitoring.

PROGRAM EDUCATIONAL OBJECTIVES

1. Lead a successful career by designing, analysing and solving various problems in the field of Computer Science & Engineering.
2. Pursue higher studies for enduring edification.
3. Exhibit professional and team building attitude along with effective communication.
4. Identify and provide solutions for sustainable environmental development.

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We wouldlike to express our deep sense of gratitude and indebtedness to each and every one who hashelped us make this project a success.

We heartily thank our Principal, Dr. MOHAN BABU G N, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We heartily thank our Professor and Head of the Department, Dr. ANIL G N, Department of Computer Science and Engineering, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We gracefully thank our Project Guide, Mr. Shankar R, Assistant Professor, Department of Computer Science and Engineering for his intangible support and forbeing constant backbone for our project.

Special thanks to all the staff members of Computer Science Department for theirhelp and kind co-operation.Lastly, we thank our parents and friends for the support and encouragement given throughout in completing this precious work successfully.

KUSHAL SR USN: 1BY18CS080

PRATEEK USN: 1BY15CS111

SANTOSH PM USN: 1BY18CS144

ABSTRACT

A steam engine is a heat engine that performs mechanical work using steam as its working fluid. Heat is obtained from fuel burnt in a closed firebox .The heat is transferred to the water in a pressurized boiler, ultimately boiling the water and transforming it into saturated steam. Steam in its saturated state is always produced at the temperature of the boiling water, which in turn depends on the steam pressure on the water surface within the boiler. The steam is transferred to the motor unit which uses it to push on pistons to power machinery. The used, cooler, lower pressure steam is exhausted to atmosphere. The engine is initially at rest. On right clicking, the user is provided with a menu which provides five options-shaded, animate, increase speed, decrease speed, transparent. The animate option starts the steam engine from rest or stops the engine if it is running. The speed of the engine can then be increased by the increase speed option or decreased by the decrease speed option. The texture of the engine can be changed by the shaded option. There are two textures. One being the normal solid fill and the other being wireframe. The transparent option makes the front portion of the cylinder transparent and shows the up and down motion of the piston. The simulation helps in understanding the working of the engine. It clearly shows how the linear motion is converted into rotary motion.

CONTENTS

INTRODUCTION TO PROJECT DESCRIPTION

The working of the steam engine is simulated by using OpenGL library functions. The linear motion of the piston inside the cylinder and how the linear motion is converted into rotary motion by the crankshaft is shown

The engine can be rotated in 3-dimensions which demonstrates the 3-dimensional rotation of an object about an axis. The lighting effect is also demonstrated using a light source. The motion of the engine also demonstrates the use of animation in OpenGL.The wireframe view of the cylinder shows how a 3-dimensional object like a cylinder and disc is approximated using line segments and polygons. 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 increments the crank angle by ANGLE_STEP, updates the head angle and notifies the system that the screen needs to be updated

H/W AND S/W REQUIREMENTS

Hardware Requirements:

- Intel P3 processor and above
- 32MB or 64MB RAM(Minimum), 128MB or 512MB for faster processing
- Minimum of 1GHz processor for better results
- Input and Output Devices

Software Requirements:

- Microsoft Visual C++ 6.0
- GL Files(GL Library Files)
- Windows or Linux or Mac OS

Good Interface(Ex:Menu,Buttons,etc)

System design

Transformation Matrices :

Vertex transformations (such as rotations, translations and scaling) and projections (such as perspective and orthographic) can all be represented by applying an appropriate 4 × 4 matrix to the coordinates representing the vertex. If v represents a homogeneous vertex and M is a 4 × 4transformation matrix, then Mv is the image of v under the transformation by M. After transformation, all transformed vertices are clipped so that *x*, *y*, and *z* are within the screen co-ordinates.

Although any nonsingular matrix M represents a valid projective transformation, a few special matrices are particularly useful. These matrices are listed below :

Translation :

The call glTranslate**(***x*, *y*, *z***)** generates T, where

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling :

The call glScale**(***x, y, z***)** generates S, where

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } S^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation :

The glRotate**()** command generates a matrix for rotation about an arbitrary axis. Often, you're rotating about one of the coordinate axes; the corresponding matrices are as follows

glRotate*$(a, 1, 0, 0)$: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

glRotate*$(a, 0, 1, 0)$: $\begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

glRotate*$(a, 0, 0, 1)$: $\begin{bmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Modeling of the Steam Engine

The cylinder, piston, flywheel, crank, crankshaft are modeled by using a cylinder. Two disks are placed on the top and bottom face of the cylinder in order to provide a more realistic look when the engine is rotated. The length and the radius of the cylinder is specified according to the dimensions of the parts of the cylinder. For example, the crank is a long rod with less thickness, hence its radius has to be less and length has to be more, whereas the flywheel is a thick large disc hence it is modeled using a cylinder of less length but more radius.

The various parts are first drawn with reference to the relative positions with respect to that part. The different parts are then assembled together by placing them in the appropriate position on the screen.The crank is the part of the engine which has no linear motion hence it is placed first.The other parts are then placed using the crank as a reference. All these transformations are carried out using the OpenGL functions glTranslate(),glRotate() and glScale()

Display Lists

Display lists may improve performance since you can use them to store OpenGL commands for later execution. It is often a good idea to cache commands in a display list if you plan to redraw the same geometry multiple times, or if you have a set of state changes that need to be applied multiple times. Using display lists, you can define the geometry and/or state changes once and execute them multiple times.

When running locally, you can often improve performance by storing frequently used commands in a display list. Some graphics hardware may store display lists in dedicated memory or may store the data in an optimized form that is more compatible with the graphics hardware or software.

The way in which the commands in a display list are optimized may vary from implementation to implementation. For example, a command as simple as glRotate*() might show a significant

improvement if it's in a display list, since the calculations to produce the rotation matrix aren't trivial (they can involve square roots and trigonometric functions). In the display list, however, only the final rotation matrix needs to be stored, so a display-list rotation command can be executed as fast as the hardware can execute glMultMatrix*(). A sophisticated OpenGL implementation might even concatenate adjacent transformation commands into a single matrix multiplication.

Suppose a transformation is to be applied to some geometric objects and then draw the result. The code may look like :

```
glNewList(1, GL_COMPILE);
draw_some_geometric_objects();
glEndList();
glLoadMatrix(M);
glCallList(1);
```

However, if the geometric objects are to be transformed in the same way each time, it is better to store the matrix in the display list. For example, if the code were to be written as follows, some implementations may be able to improve performance by transforming the objects when they are defined instead of each time they are drawn:

```
glNewList(1, GL_COMPILE);
glLoadMatrix(M);
draw_some_geometric_objects();
glEndList(); glCallList(1);
```

Quadrics

The base OpenGL library only provides support for modeling and rendering simple points, lines, and convex filled polygons. Neither 3D objects, nor commonly used 2D objects such as circles, are directly available.

The GLU also provides routines to model and render tessellated, polygonal approximations for a variety of 2D and 3D shapes (spheres, cylinders, disks, and parts of disks), which can be calculated with quadric equations. This includes routines to draw the quadric surfaces in a variety of styles and orientations.

To use a quadrics object, follow these steps.

1. To create a quadrics object, use gluNewQuadric**()**.

2. Specify the rendering attributes for the quadrics object .

   2.1. Use gluQuadricOrientation**()** to control the winding direction and differentiate the interior from the exterior.

   2.2. Use gluQuadricDrawStyle**()** to choose between rendering the object as points, lines, or filled polygons.

   2.3. For lit quadrics objects, use gluQuadricNormals**()** to specify one normal per vertex or one normal per face. The default is that no normals are generated at all.

   2.4. For textured quadrics objects, use gluQuadricTexture**()** if you want to generate texture coordinates.

3. Prepare for problems by registering an error-handling routine with gluQuadricCallback**()**.

Then, if an error occurs during rendering, the routine you've specified is invoked.

4. Now invoke the rendering routine for the desired type of quadrics object: gluSphere**()**, gluCylinder**()**, gluDisk**()**, or gluPartialDisk**()**. For best performance for static data, encapsulate the quadrics object in a display list.

5. When you're completely finished with it, destroy this object with gluDeleteQuadric**()**. If you need to create another quadric, it's best to reuse your quadrics object.

Animating the Engine

Once the engine is modeled and drawn on the screen the next step is to animate the engine. In order to achieve this the amount of rotation of the cylinder, piston, crank, crankbell and flywheel is to be determined. The angle by which the crank, flywheel, crankbell are to be rotated is initially set to 0.It is then incremented in steps of 5.The angle of rotation of the cylinder depends on the rotation of the crank. The rotation of the cylinder head for different crank angles is calculated using the formula:

MAGNITUDE*atan((ARC_RADIUS * sin (PHASE – (k/FREQ_DIV)) /

(ARC_LENGTH – (ARC_RADIUS*cos(PHASE – (k/FREQ_DIV))))

for the values of k from 0 to 360 degrees and stored in an array. Then the rotation of the cylinder for a particular crank angle is obtained by indexing the crank angle into the array. The piston connects the crankbell to the cylinder hence it should be rotated by an angle equal to the difference between the crank angle and the head angle.The rotations are computed using the

glRotate() function and the model is redisplayed after each step.This gives the perception of the animation.

Light Effects

The OpenGL lighting model considers the lighting to be divided into four independent components: emissive, ambient, diffuse, and specular. All four components are computed independently and then added together.

Ambient illumination is light that's been scattered so much by the environment that its direction is impossible to determine - it seems to come from all directions.When ambient light strikes a surface, it's scattered equally in all directions. The diffuse component is the light that comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the surface. Once it hits a surface, however, it's scattered equally in all directions, so it appears equally bright, no matter where the eye is located. specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction.Shiny metal or plastic has a high specular component, and chalk or carpet has almost none.

APIs WITH THEIR DESCRIPTION

Attributes

void glColor*(TYPE r,TYPE g,TYPE b,TYPE a)

sets the present RGBA colors. The maximum and minimum values of the types are 1.0 and 0.0 respectively

void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)

sets the present RGBA clear color used when clearing the color buffer.Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

Working with Window system

void GLflush()

Forces any buffered OpenGL command to execute

void glutInit(int argc, char **argv);

glutInit**()** should be called before any other GLUT routine, because it initializes the GLUT library. glutInit**()** will also process command line options, but the specific options are window system dependent. For the X Window System, -iconic, -geometry, and -display are examples of command line options, processed by glutInit**()**. (The parameters to the glutInit**()** should be the same as those to main**()**.

void glutInitDisplayMode(unsigned int mode);

Specifies a display mode (such as RGBA or color-index, or single- or double-buffered) for windows created when glutCreateWindow**()** is called. You can also specify that the window have an associated depth, stencil, and/or accumulation buffer. The mask argument is a bitwise ORed combination of GLUT_RGBA or GLUT_INDEX, GLUT_SINGLE or

GLUT_DOUBLE, and any of the buffer-enabling flags: GLUT_DEPTH, GLUT_STENCIL, or GLUT_ACCUM. For example, for a double-buffered, RGBA-mode window with a depth and stencil buffer, use GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH | GLUT_STENCIL. The default value is GLUT_RGBA | GLUT_SINGLE (an RGBA, single-buffered window).

void glutInitWindowSize(int width, int height);

void glutInitWindowPosition(int x, int y);

Requests windows created by glutCreateWindow**()** to have an initial size and position. The

arguments (x, y) indicate the location of a corner of the window, relative to the entire display.The width and height indicate the window's size (in pixels). The initial window size and position are hints and may be overridden by other requests.

int glutCreateWindow(char *name);

Opens a window with previously set characteristics (display mode, width, height, and so on).

The string name may appear in the title bar if your window system does that sort of thing. The window is not initially displayed until glutMainLoop**()** is entered, so do not render into the window until then.The value returned is a unique integer identifier for the window. This identifier can be used for controlling and rendering to multiple windows (each with an OpenGL rendering context)from the same application.

void glutPostRedisplay()

Requests that the display callback be executed after the current callback returns.

void glutSwapBuffers()

Swaps the front and back buffers.

Callback Functions

void glutDisplayFunc(void (*func)(void))

registers the display function func that is executed when the window needs to be redrawn.

void glutReshapeFunc(void *f(int width,int height))

Registers the reshape callback function f.The callback function returns the height and width of the new window.the reshape callback invokes the display callback.

void glutKeyboardFunc(void (* func)(unsigned int key, int x, int y);

Specifies the function, func, that's called when a key that generates an ASCII character is pressed. The key callback parameter is the generated ASCII value. The x and y callback parameters indicate the location of the mouse (in window-relative coordinates) when the key was pressed.

void glutMouseFunc(void (* func)(int button, int state, int x, int y));

Specifies the function, func, that's called when a mouse button is pressed or released. The button callback parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON. The state callback parameter is either GLUT_UP or GLUT_DOWN, depending upon whether the mouse has been released or pressed. The x and y callback parameters indicate the location (in window-relative coordinates) of the mouse when the event occurred.

Enabling Features

void glEnable(GLenum feature)

enables an OpenGL feature.Features that can be enabled include GL_DEPTH_TEST, GL_LIGHTING, GLLIGHTi, GL_LINE_SMOOTH.

IMPLEMENTATION

ALGORITHM

Drawing Cylinder

1. Specify the radius of top face and bottom face.
2. Draw a cylinder using glucylinder () function .
3. Set radius of inner and outer circles of disc.
4. Draw a disc at bottom face and top using gludisc().
5. Rotate disc by 180 degree about y axis.

Drawing Cylinder Head of the Engine

1. Set the color.
2. Draw cylinder.
3. Rotate by 90 degree about x-axis.
4. Draw disc at the top face.
5. Rotate by head angle about the x-axis at pivot point.
6. Rotate disc by 180 degree about x-axis.

Drawing Crank of the Engine

1. Draw Cylinder
2. Rotate it by Crank angle about X-axis
3. Rotate it by 90 degree about Y-axis.

Drawing Crankbell of the Engine

1. Draw Cylinder
2. Rotate it by crank angle about X-axis
3. Rotate it by 90 degree about Y-axis
4. Draw a smaller cylinder to connect the crank bell and piston
5. Rotate it by crank angle about X-axis
6. Rotate it by 90 degree about Y-axis.

Drawing the Piston of the Engine

1. Draw a tiny Cylinder to connect the piston to the crankbell.
2. Rotate it by 180 degree about Y-axis.
3. Rotate it by crank angle-head angle about X-axis
4. Rotate it by 90 degree about Y-axis
5. Draw a Cylinder representing the piston rod.
6. Rotate it by 180 degree about Y-axis
7. Rotate it by crank angle-head angle about X-axis
8. Rotate it by -90 degree about Y-axis

9. Draw Cylinder representing the main part of the piston
10. Rotate it by 180 degree about Y-axis
11. Rotate it by crank angle-head angle about X-axis
12. Rotate it by -90 degree about Y-axis

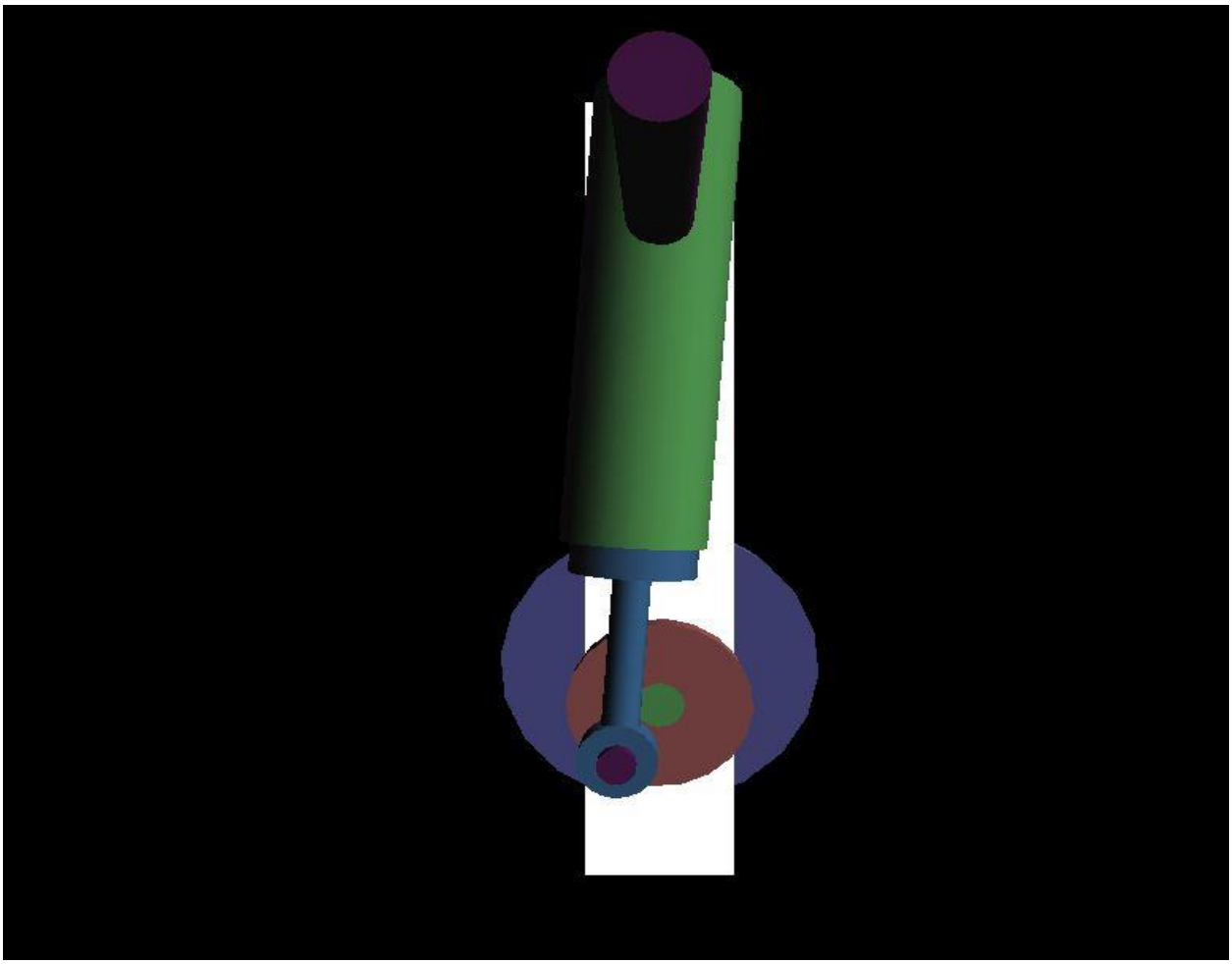
Drawing the Flywheel of the Engine

1. Draw Cylinder representing the Flywheel of the Engine
2. Rotate it by crank angle about X-axis
3. Rotate it by 90 degree about Y-axis

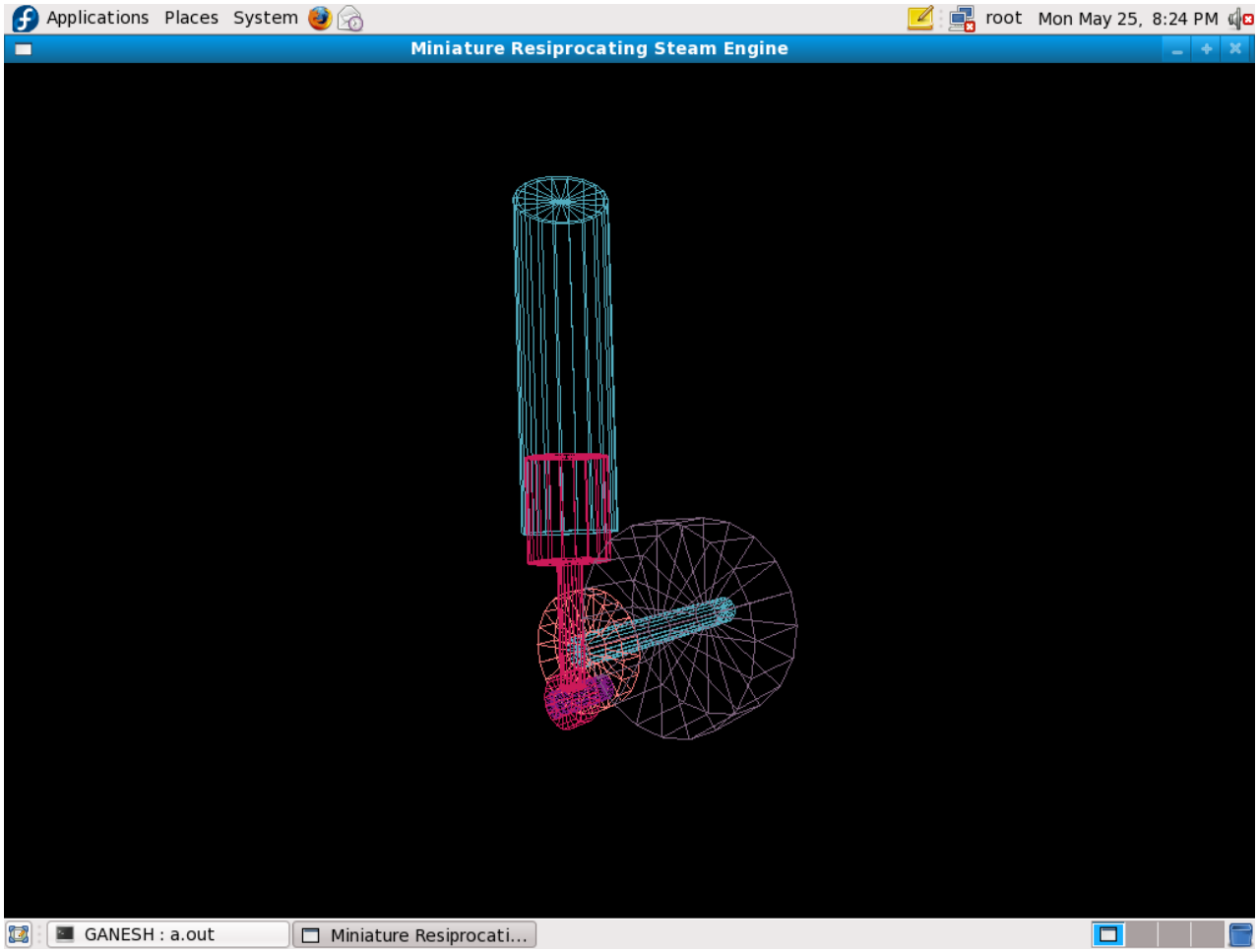
RESULTS AND DISCUSSION

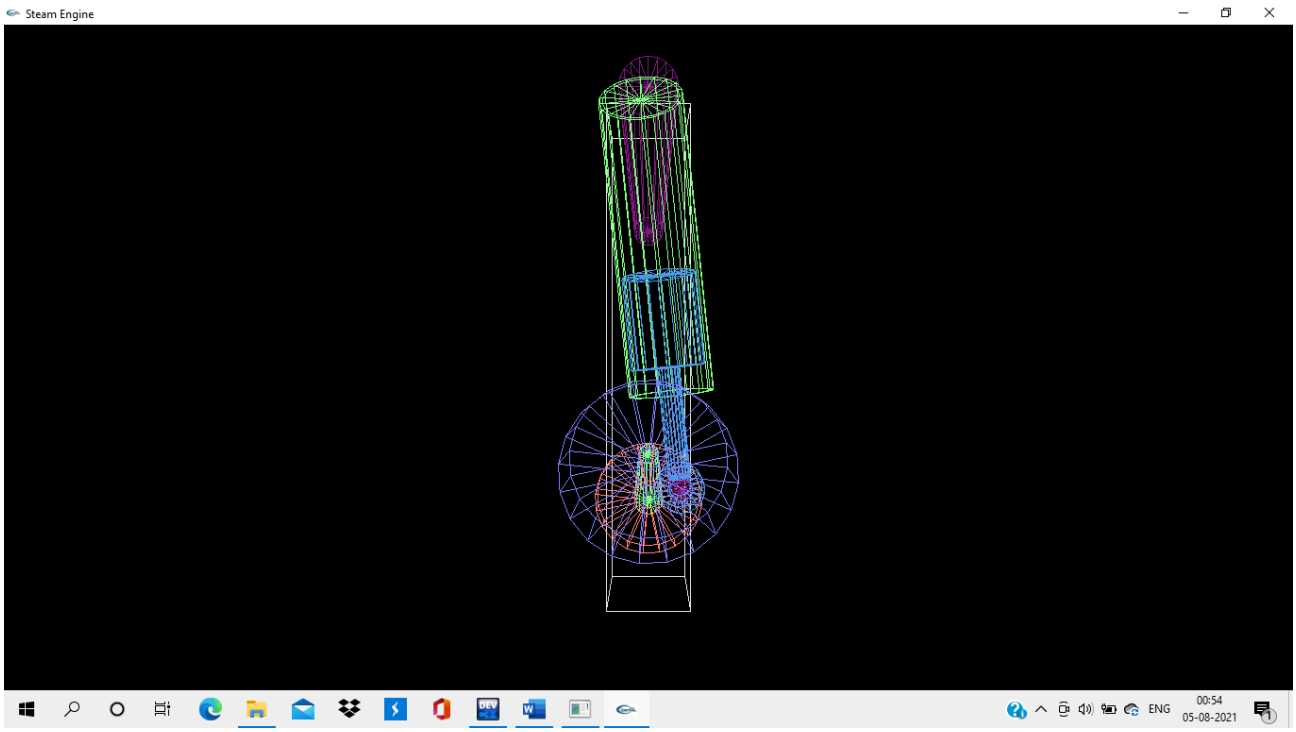
Screenshots

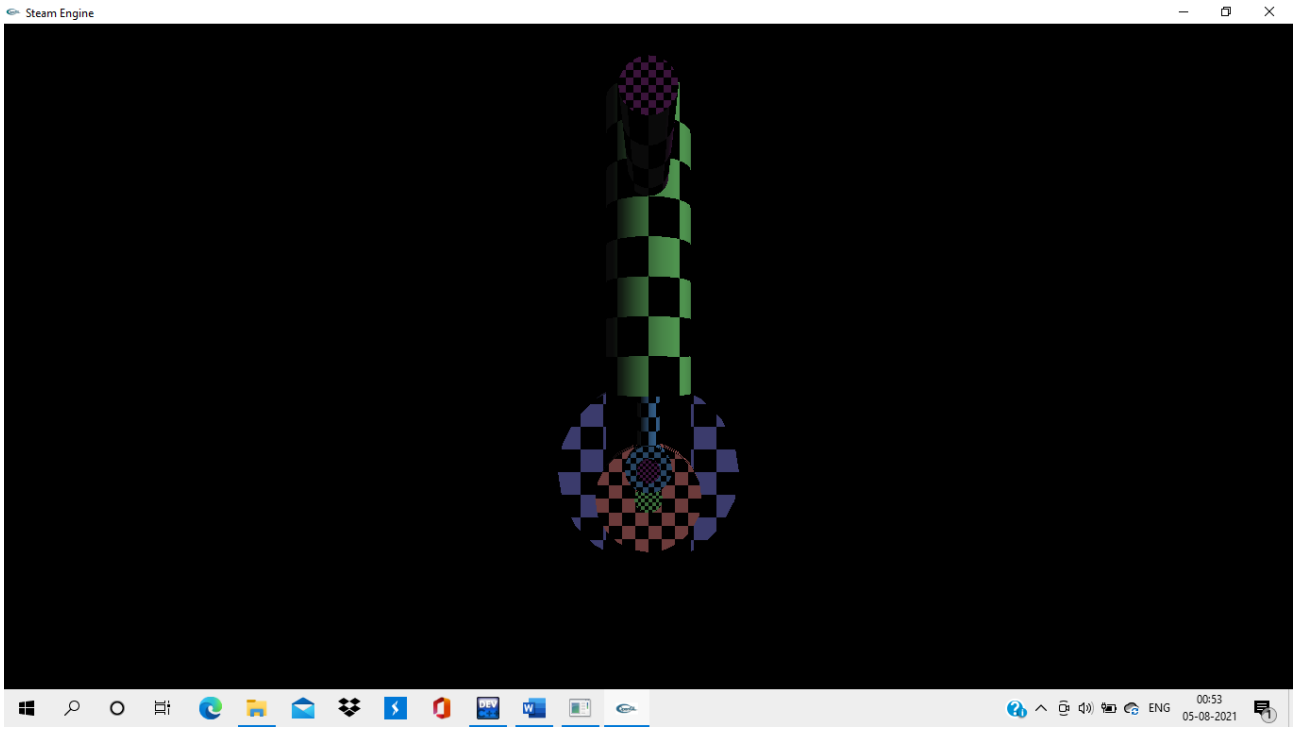
1.Screenshot showing the initial shaded model of the engine



2. Screen shot showing the wireframe model of the engine which is in motion and being rotated about the Y-axis




Steam Engine
ENG
00:54
05-08-2021



Description: The program simulate the working of miniature steam engine with wire frame and 3D shaped views . The animation shows the movement as how the steam engine work. It is quite complex program make use of texture and simple check pattern. It has a wide range of user interaction as well.

Usages: Click either button and choose to animate and select the views and exit.

Shaded : For shading the object.
Animation : To start animating the work.
Texture : Texture mapping.
Transparency : Make use of transparency in objects.
Right Light : To view as light falling on right side.
Left Light : To view as light falling on left side.
Speed UP : Speed up the motion by 1.
Slow Down : Speed down the motion by 1.

Keyboard functions:
a: Animate manually in anti-clockwise direction.
z: Animate manually in clockwise direction.
s: Toggle b/w wire frame and shaped views.
+: Increase the speed by 1.
-: Decrease the speed by 1.
2: Moving the whole engine down.
4: Moving the whole engine left.
6: Moving the whole engine right.
8: Moving the whole engine up.

CONCLUSION

The mini project develop as a scope for future enhancement too as follows . it can be used in CAD and architecture field. It can be used in gaming. This can be used in vedio imaging and creating animated movies. It can be tired to make more interactive like a user want a particular details of project is shown pictorially which helps in commercial and business world.

REFERENCES :

- Edward's Angel's interactive computer graphics Pearson education 5th edition.

WEBSITES :

- www.OpenGL Redbook.
- www.OpenGL simple examples.
- www.OpenGL programming guide.