1.INTRODUCTION

1.1 Introduction to Computer Graphics

Graphics is defined as any sketch or a drawing or a special network that pictorially represents some meaningful information. Computer Graphics is used where a set of images needs to be manipulated or the creation of image in the form of pixels and is drawn on the computer. Computer Graphics can be used in digital photography, film, entertainment, electronic gadgets, and all other required core technologies. It is a vast subject and area in the field of computer science. Computer Graphics can be used in UI design, rendering, geometric objects, animation, and many more. In most areas, computer graphics is an abbreviation of CG. There are several tools used for the implementation of Computer Graphics.

Computer graphics started with the display of data on hardcopy plotters and Cathode Ray Tube (CRT) screens soon after the Introduction of computers themselves. It has grown to include the Creation, Storage, and Manipulation of Models and Images of objects. These models come from a diverse and expanding set of fields and include physical, mathematical, engineering, architectural, and even conceptual structures, natural phenomenon, and so on. Computer graphics today is largely interactive: the user controls the contents, structure, and appearance of objects and of their displayed images by using input devices, such as a keyboard, mouse, or touch-sensitive panel on the screen. Because of the close relationship between the input devices and the display, the handling of such devices is included in the study of computer graphics.

1.2 Uses of Computer Graphics:

Computer graphics are used in many different areas of industry, business, government, education, entertainment, etc.

User Interfaces

Word-processing, spreadsheet, and desktop-publishing programs are typical applications of such user-interface techniques.

Interactive Plotting in Business, Science, and Technology

The common use of graphics is to create 2D and 3D graphs of mathematical, physical and economic functions, histograms, and bar and pie charts.

Computer-Aided Drafting and Design (CAD)

In CAD, interactive graphics are used to design components and systems of mechanical, electrical, and electronic devices including structures such as buildings, automobile bodies, airplanes, ship hulls, etc.

Simulation and Animation for Scientific Visualization and Entertainment

Computer-produced animated movies are becoming increasingly popular for scientific and engineering visualization. Cartoon characters will increasingly be modelled on the computer as 3D shape descriptions whose movements are controlled by computer commands.

2D Graphics

These editors are used to draw 2D pictures (line, rectangle, circle, and ellipse) and alter those with operations like cut, copy and paste. These may also support features like translation, rotation, etc.

3D Graphics

These editors are used to draw 3D pictures (line, rectangle, circle, and ellipse). These may also support features like translation, rotation, etc.

1.3 Introduction to OpenGL

Interface

OpenGL is an application program interface (API) offering various functions to implement primitives, models, and images. This offers functions to create and manipulate render lighting,coloring, a viewing of the models. OpenGL offers different coordinate systems and frames. OpenGL offers translation, rotation, and scaling of objects. Functions in the main GL library have names that begin with gl and are stored in a library usually referred to as GL. The secondis the OpenGL Utility Library (GLU). The library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letter glu. Rather than using a different library for each system, we use a readilyavailable library called OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

Overview

- OpenGL (Open Graphics Library) is the interface between a graphics program and graphics hardware. It is streamlined. In other words, it provides low-level functionality. For example, all objects are built from points, lines, and convex polygons. Higher-level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program, there is a current model transformation.

- It is a rendering pipeline. The rendering pipeline consists of the following steps:

* Defines objects mathematically.
* Arranges objects in space relative to a viewpoint.
* Calculates the color of the objects.

1.4 Introduction to Project-title:

This Project is on **" BIKE RIDING"** Computer Graphics using *OpenGL Functions.* It is a User interactive program where in the User can view the required display by making use of the input devices like Keyboard and Mouse. This project mainly consists of a bike with a street. This street is filled with few street lights ,this game has few features , where a bike can do a wheelie , increase its speed by doing a upshift , and also has downshift feature in this bike , with the help of specified keyword bike can also do a wheelie .

.

1. 'I' or 'i' - > To increase the gear (upshift).
2. 'D'or 'd' - > To decrease the gear(downshift).
3. 'W' or 'w' - > To start the bike.
4. 'U' or 'u' - > Perform wheelie.
5. 'S' or 's' - > To stop the bike.
6. 'ESC' - > Exit

2. OpenGL Architecture

2.1 Pipeline Architectures

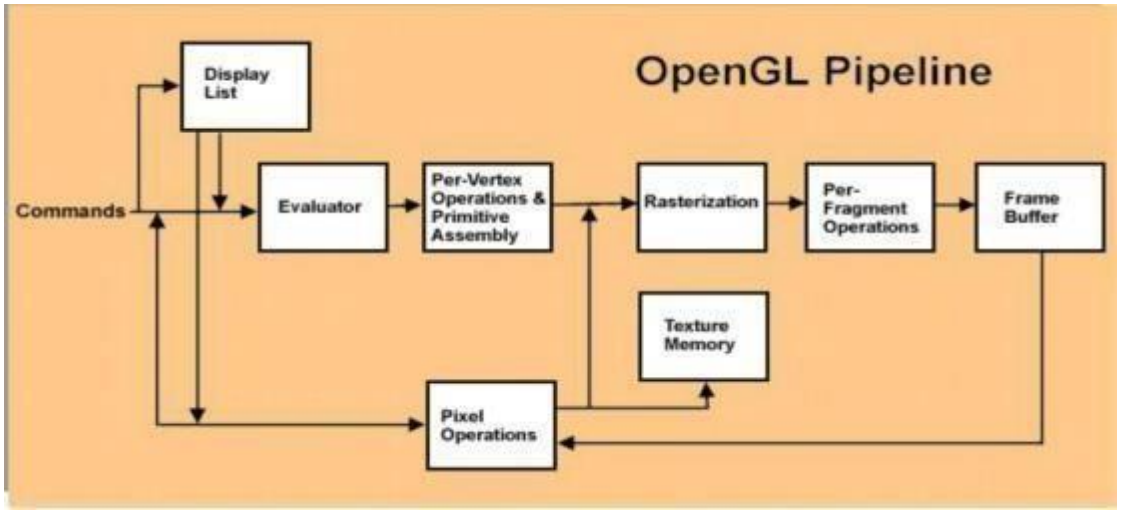


FIG:2.1 OPENGL PIPELINE ARCHITECTURE

Display Lists: All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. (1 alternative to retaining data in a display list is processing the data immediately - also known as immediate mode.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators: All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive thevertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colours,and spatial coordinate values from the control points.

Per-Vertex Operations: For vertex data, next is the "per-vertex operations" stage, which converts. The vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.If advanced features are enabled, this stage is even busier. If texturing is used, texturecoordinates may be generated and transformed here. If lighting is enabled, the lightingcalculations are performed using the transformed vertex, surface normal, light sourceposition, material properties, and other lighting information to produce a colour value.

Primitive Assembly: Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane.Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases,this is followed by perspective division, which

makes distant geometric objects appearsmaller than closer objects. Then view port and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected bya culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results or this stage are complete geometric primitives, which are the transformed and clipped vertices with related colour, depth, and sometimes texture- coordinate values and guidelines for the rasterization step.

Pixel Operations: While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number ofcomponents. Next the data is scaled, biased, and processed by a pixel map. The resultsare clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias,mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixelcopy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

Texture Assembly: An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images areused, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to acceleratetexture performance. There may be specialized, high-performance texture memory. Ifthis memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

Rasterization: Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line andpolygon stipples, line width, point size, shading model, and coverage calculations to support ant aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

2.2 OpenGL Engine and Drivers

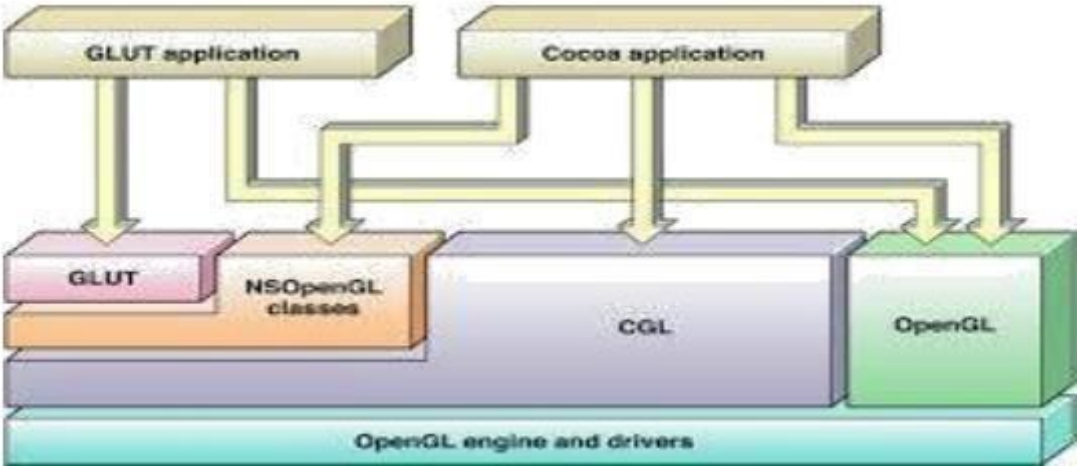


FIG:2.2 OPENGL ENGINE AND DRIVERS

CPU-GPU Cooperation

The architecture of OpenGL is based on a client-server model. An application program writtento use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine (including the GLSL shader programs we write) is the "server" andruns on the GPU. Geometry and many other types of attributes are stored in buffers called Vertex Buffer Objects (or VBOs). These buffers are allocated on the GPU and filled by yourCPU program. We will get our first glimpse into this process (including how these buffers areallocated, used, and deleted) in the first sample program we will study.

Modeling, rendering, and interaction is very much a cooperative process between the CPU client program and the GPU server programs written in GLSL. An important part of the design process is to decide how best to divide the work and how best to package and communicate required information from the CPU to the GPU. There is no standard "best way" to do this that is applicable to all programs, but we will study a few very common approaches.

Window Manager Interfaces

OpenGL is a pure output-oriented modeling and rendering API. It has no facilities for creating and managing windows, obtaining runtime events, or any other such window system dependent operation. OpenGL implicitly assumes a window-system interface that fills these needs and invokes user-written event handlers as appropriate.

It is beyond the scope of these notes to list, let alone compare and contrast, all window manager interfaces that can be used with OpenGL. Two very common window system interfaces are:

GLFW: Runs on Linux, Macintosh, and Windows.

GLUT: (actually *freeglut*): A window interface that used to be the most common one used when teaching OpenGL. It, too, runs on Linux, Macintosh, and Windows.

2.3 Application Development-API's

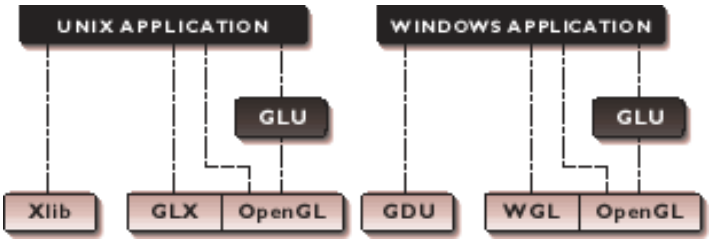


FIG:2.3 APPLICATIONS DEVELOPMENT(API'S)

This diagram demonstrates the relationship between OpenGL GLU and windowing APIs.
Leading software developers use OpenGL, with its robust rendering libraries, as the 2D/3D graphics foundation for higher-level APIs. Developers leverage the capabilities of OpenGL to deliver highly differentiated, yet widely supported vertical market solutions. For example,Open Inventor provides a cross-platform user interface and flexible scene graph that makes iteasy to create OpenGL applications. IRIS Performer < leverages OpenGL functionality and delivers additional features tailored for the demanding high frame rate markets such as visualsimulation and virtual sets OpenGL Optimizer is a toolkit for real-time interaction, modification, and rendering of complex surface-based models such as those found in CAD/CAM and special effects creation. OpenGL Volumizer is a high-level immediate modevolume rendering API for the energy, medical and sciences markets. OpenGL Shader providesa common interface to support realistic visual effects, bump mapping, multiple textures, environment maps, volume shading and an unlimited array of new effects using hardware acceleration on standard OpenGL graphics cards.

3. Implementation

Code section:

```
#include<windows.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
#include<GL/glut.h>
#include<stdio.h>
#include<conio.h>

int theta =0 ,n,j,axis=-1,t=10000,x=1,s=1,ax=15,wflag=0,gear=1,submenu;
double i;
float a=-175.0,ac=.50,b=195.0,x1=-195,x2=-155,x3=-175,x4=-200,x5=-150,x6=-
170,x7=-180,x8=0,x9=20,cl1=1.0,cl2=0.0,cl3=0.0;
float d=0,dir=-1,st=0;

void wheels()        //Function to draw wheels
{
glLineWidth(2.5);
glPointSize(2.0);
glColor3f(0,0,0);
glBegin(GL_LINES);
glVertex3f(5,0,1);
glVertex3f(-5,0,10);
glVertex3f(0,5,10);
glVertex3f(0,-5,10);
glEnd();


glBegin(GL_POINTS);
glVertex2f(0,0);
glEnd();

glBegin(GL_POINTS);
for(j=5;j<9;j++)
for(i=0;i<360;i++)
{
glVertex3f(j*cos(i),j*sin(i),10);
}
glEnd();
}
```

```
void body()
{
glColor3f(0.9,0.9,0.5); //headlight
glBegin(GL_POLYGON);
glVertex2f(28.0,14.0);
glVertex2f(35.0,15.0);
glVertex2f(34.0,25.0);
glVertex2f(27.0,27.0);
glEnd();

glColor3f(0.8,0.6,0.9);     //tank
glBegin(GL_POLYGON);
glVertex2f(13.0,22.5);
glVertex2f(26.0,22.5);
glVertex2f(30.0,10.0);
glVertex2f(10.0,10.0);
glEnd();

glColor3f(0,0,0);    //tail
glBegin(GL_TRIANGLES);
glVertex2f(-9.0,22.5);
glVertex2f(3.5,21.0);
glVertex2f(8.0,8.0);
glEnd();
glColor3f(1,0,0);    //centre part
glBegin(GL_POLYGON);
glVertex2f(10.0,-3.0);
glVertex2f(20.0,-3.0);
glVertex2f(35.0,15.0);
glVertex2f(0.0,17.5);
glEnd();
}

void geaar()
{
char *p="GEAR";
glColor3f(0.0,0.0,0.0);
for(int v=0;v<strlen(p);v++)
{
glRasterPos2i(40+(10*v),180);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,p[v]);
}
```

```
glColor3f(1.0,1.0,1.0);
if(gear==4)
{
glRasterPos2i(90,180);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'4');
}
else if(gear==3)
{
glRasterPos2i(90,180);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'3');
}
 else if(gear==2)
{
glRasterPos2i(90,180);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'2');
}
  else
{
glRasterPos2i(90,180);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,'1');
}
}

void road()
{
glColor3f(0.9,0.9,0.9); //Road Allignment
for(int x=-500;x<500;x=x+40)
{
glBegin(GL_POLYGON);
glVertex3f(x,-103,0);
glVertex3f(x+15,-103,0);
glVertex3f(x+15,-104,0);
glVertex3f(x,-104,0);
glEnd();
}

glColor3f(0.4,0.4,0.4);   //Road
glBegin(GL_POLYGON);
glVertex3f(500,-90,0);
glVertex3f(-500,-90,0);
glVertex3f(-500,-116,0);
glVertex3f(500,-116,0);
glEnd();
```

```
glColor3f(0.6,0.6,1.0);   //Sky
glBegin(GL_POLYGON);
glVertex3f(-500,200,0);
glVertex3f(500,200,0);
glVertex3f(500,-103,0);
glVertex3f(-500,-103,0);
glEnd();

glColor3f(0.0,0.6,0.0); //Grassland
glBegin(GL_POLYGON);
glVertex3f(-500,-110,0);
glVertex3f(500,-110,0);
glVertex3f(500,-200,0);
glVertex3f(-500,-200,0);
glEnd();

glColor3f(0,0,0);            //Lamp
for(int y=-400;y<400;y=y+150)
{
glLineWidth(5.0);
glBegin(GL_LINES);
glVertex3f(y,-116,1);
glVertex3f(y,10,1);
glVertex3f(y,10,1);
glVertex3f(y+20,10,1);
glEnd();
glPointSize(15.0);

glBegin(GL_POINTS);   //Lamp Light
glVertex3f(y+20,10,1);
glEnd();
}
}


void display()        //Function to display every thing and provide rotation
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glPushMatrix();
if(a>=220.0)
a=-220;
glTranslatef(a,-92,0);       //move front wheels
glRotatef(theta,0.0,0.0,1.0);
wheels();
```

```
glPopMatrix();
glPushMatrix();     //move rear wheel
glTranslatef(a+30,-92,0);
glRotatef(theta,0.0,0.0,1.0);
wheels();
glPopMatrix();

glPushMatrix();     //move body
glTranslatef(a,-92,0);
body();
glPopMatrix();

if(wflag==1)
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
geaar();
glPushMatrix();
glTranslatef(a,-92,0);
glRotatef(theta,0.0,0.0,1.0);
wheels();
glPopMatrix();

glPushMatrix();
glTranslatef(a+24,-61,0);
wheels();
glPopMatrix();
glPushMatrix();
glTranslatef(a+1,-91,0);
glRotatef(46.0,0.0,0.0,1.0);
body();
glPopMatrix();
}

geaar();
glPushMatrix();     //move road
if(b<=-300.0)
b=+300;
glTranslatef(b,0,0);
road();
glPopMatrix();
glFlush();
glutSwapBuffers();
glutPostRedisplay();
}
```

```
void spin()
{
theta+=ax*axis;
if(theta>360)
theta=0;
glutPostRedisplay();
axis=0;
}

void mytimer(int v)
{
glutTimerFunc(t/60,mytimer,v);
spin();

}

void my_menu(int id)
{
switch(id)
{
case 7:exit(0);
break;
}
}


void color_menu(int id)
{
switch(id)
{
case 1:cl1=0.0, cl2=0.0, cl3=0.0;
break;
case 2:cl1=0.0, cl2=0.0, cl3=0.8;
break;
case 3:cl1=0.0, cl2=0.9, cl3=0.0;
break;
case 4:cl1=1.0, cl2=0.0, cl3=0.0;
break;
}
}

void menu()
{
```

```
submenu=glutCreateMenu(color_menu);
glutAddMenuEntry("Black",1);
glutAddMenuEntry("Blue",2);
glutAddMenuEntry("Green",3);
glutAddMenuEntry("Red",4);
glutCreateMenu(my_menu);
glutAddSubMenu("Color",submenu);
glutAddMenuEntry("Exit",7);

glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void keys(unsigned char key,int x,int y)
{
if(key=='x' || key=='X')
{
a--;
}
if(key=='U'||key=='u')
{
wflag=1;
}
if(key=='s'||key=='S')
{
wflag=0;
}

if(key=='I'  || key=='i')
{
ax=ax+10;
gear++;
if(ax>=45||gear>=4)
{
ax=45;
gear=4;
}
t/=10;
d+=0.15;
}
if(key=='D' || key=='d')
{
ax=ax-10;
gear--;
if(ax<=15||gear<=1)
```

```
{
gear=1;
ax=15;
}
t*=10;
d-=0.15;
}

if(key==87 || key==119)
{
t=10;
d+=0.15;
a=a+ac;
b=b-ac;
axis=-1;
if(gear==1)
ac=.50;
else if(gear==2)
ac=1;
else if(gear==3)
ac=2;
else
ac=3;
}
if(key==27)
exit(0);
}


int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(1280,800);
glutInitWindowPosition(0,0);
glutCreateWindow("Bike Ride");

glEnable(GL_DEPTH_TEST);

printf("\nKEY USAGE : \n\n");
printf("-_____\n");
printf("I or i -> Increase Gear\n");
printf("D or d -> Decrease Gear\n");
printf("W or w -> Start Bike\n");
```

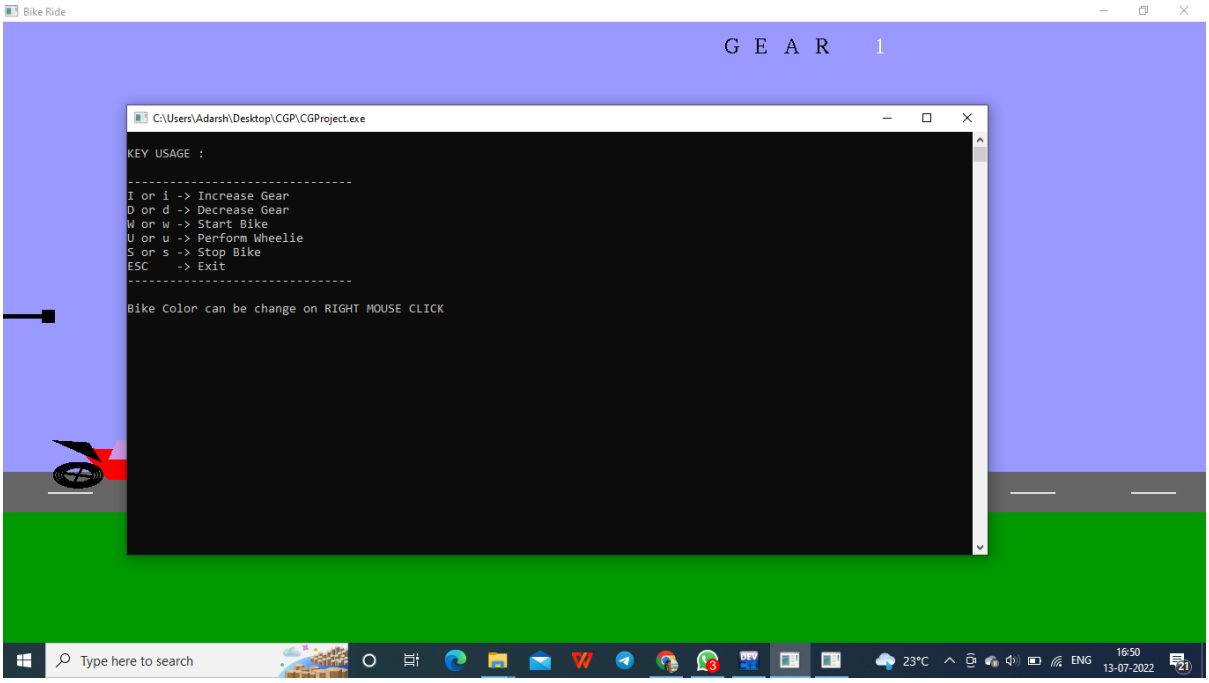
```
printf("U or u -> Perform Wheelie\n");
printf("S or s -> Stop Bike\n");
printf("ESC    -> Exit \n");
printf("-_____\n");
printf("\nBike Color can be change on RIGHT MOUSE CLICK");
glutTimerFunc(100,mytimer,60);
int flag=0;
glutKeyboardFunc(keys);
menu();
glOrtho(-200,200,-200,200,-10,10);
glClearColor(1,1,1,1);
glutDisplayFunc(display);
glutMainLoop();
}
```

Commonly used Functions of OpenGL includes:

glBegin, glEnd	The glBegin and glEnd functions delimit the verticesof a primitive or a group of like primitives.
glClear	The glClear function clearsbuffers to preset values.
glColor	These functions set the currentcolor
glFlush	The glFlush function forces the execution of OpenGLfunctions in finite time.
glLoadIdentity	The glLoadIdentity function replaces the current matrix with the identity matrix.
glMatrixMode	The glMatrixMode function specifies ch matrix is the current matrix.
glVertex	These functions specify a vertex.
gluOrtho2D	The gluOrtho2D function defines a 2-Dorthographic projection matrix.
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)
void glutMainLoop()	Cause the program to enter an event –processing loop.it should be the statement in main.

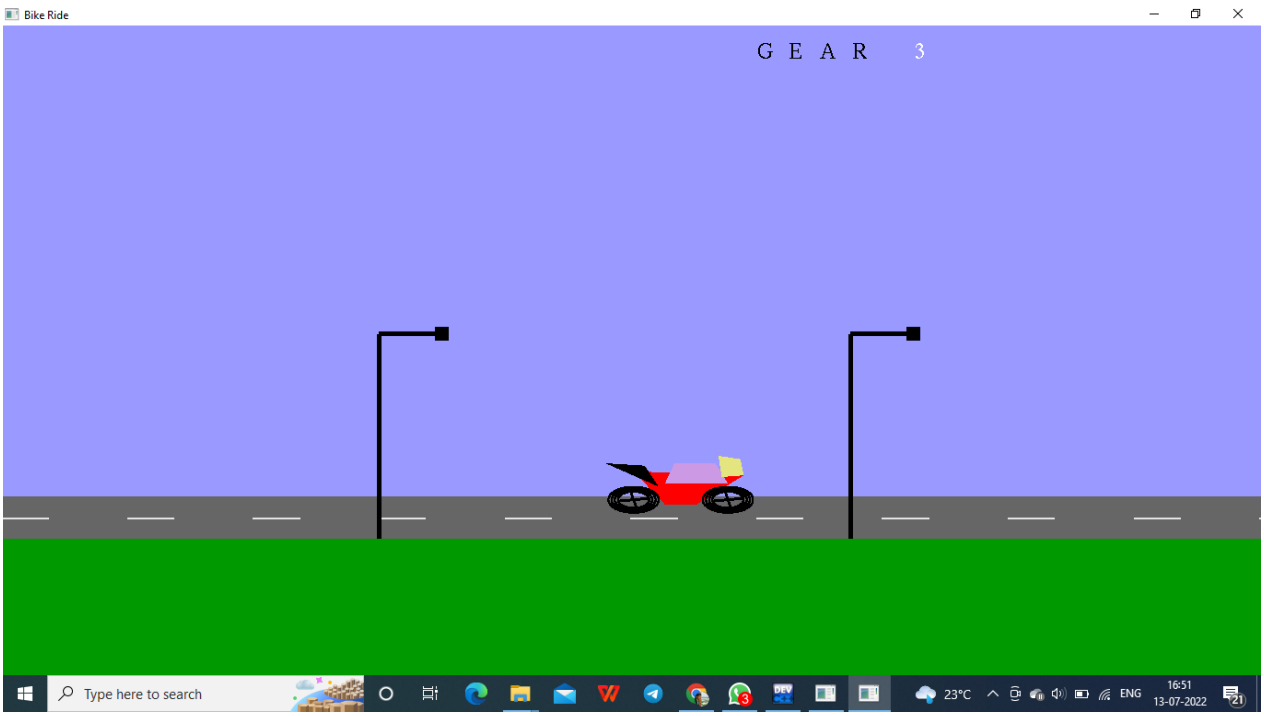
Result and Snapshots

Instruction View:



Initial View:

Upshift Gear View:



Wheeling View:

Conclusion

General Constraints:

- As the software is being built to run on windows platform, which gives access to limited conventional memory, the efficient use of the memory is very important.
- As the program needs to be run even on low-end machines the code should be efficient and optimal with the minimal redundancies.
- Needless to say, the computation of algorithms should also be robust and fast.
- It is built assuming that the standard output device (monitor) supports colors.

Assumptions and Dependencies:

- One of the assumptions made in the program is that the required libraries like GL, GLU and glut have been included.
- The user's system is required to have the C compiler of the appropriate version.
- The system is also expected to have a keyboard and mouse connected since we provide the inputs via these devices.

Web References:

- www.opengl.org
- www.google.com
- www.sourcecode.com
- www.pearsoned.co.in
- www.wikipedia.org