**PROJECT DESCRIPTION**

Open Graphics Library (Open GL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Silicon Graphics Inc., (SGI) started developing OpenGL in 1991 and released it in January 1992 applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games. Since 2006 OpenGL has been managed by the non-profit technology consortium Khronos Group.

This project is made with help of open gl we can see car with different perspective, we can start or stop the car

switch with two perspective back and front.

**Function Description**

***glutInit()***

**FGAPI void FGAPIENTRY** glutInit( **int**\* pargc,**char**\*\*argv );

Initializes GLUT and specifies command line options for the window system in use.

***GlutDisplayModel()***

**FGAPI void FGAPIENTRY** glutInitDisplayMode( **int** displayMode );

Various options for the window are set.The default is RGB color mode with single buffering.

***GlutInitWindowPosition()***

FGAPI void FGAPIENTRY glutInitWindowPosition( int x, int y );

Sets position of windows according to passed parameter.

***GlutInitWindowSize()***FGAPI void FGAPIENTRY glutInitWindowSize( int width, int height );  
Set size of windows according to passed parameter.

***GlutCreateWindow()*FGAPI int FGAPIENTRY** glutCreateWindow( **const char**\* title );   
Creates window with title passes as string or array of character.

***GlutDisplayFunc()*FGAPI void FGAPIENTRY** glutDisplayFunc( **void** (\* callback)( **void** ) );  
This function is the event handler that generates the new display whenever the display event is received.

***GlutKeyboardFunc()*FGAPI void FGAPIENTRY** glutKeyboardFunc( **void** (\* callback)( **unsigned char**, **int**, **int** ) );  
glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback

***glutReshapeFunc()***  
**FGAPI void FGAPIENTRY** glutReshapeFunc( **void** (\* callback)( **int**, **int** ) );  
.glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped.

***GlutMainLoop()*FGAPI void FGAPIENTRY** glutMainLoop( **void** );  
glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

*P.D.E.A’s*

**ANNASAHEB MAGAR COLLEGE**

# Project report

# On

**Car with perspective.**

**By**

**Aniket Undre**

**Siddhesh Sukhadeo Kand**

T.Y.B.S.C (Computer Science)

2017-2018

Under the Guidance of:-

Mrs. mam

**P.D.E.A’s**

**Annasaheb Magar Mahavidyalaya**

Hadapsar, Pune-28

Department of Computer Science

CERTIFICATE

This is to certify that **Mr Siddhesh Sukhadeo Kand , Mr Aniket Undre** have successfully and satisfactorily completed and submitted their project on “ **Car with Prespective**” ,in partial fulfillment of T.Y.B.S.C (Computer Science) submitted to **SavitribaiPhulePuneUniversity** of Pune during the Academic year 2017-2018.

Project Guide  Head of Department

(Prof. )(Dr.Mulay P.P.)

INTERNAL EXAMINER EXTERNAL EXAMINER

**Code**

#include**<GL/glut.h>**

#include**<math.h>**

#include**<stdio.h>**

#define **c** (3.14/180)

#define **PI** 3.14

#define **TWO\_PI** (2.0 \* **PI**)

#define **RAD\_TO\_DEG** (180.0 / **PI**)

*//Coordinates for the chassis of the car*

**float** p[] = {5.5, -2.5, 1}, q[] = {5.5, -7.5, 1}, r[] = {10.7, -7.5, 1}, s[] = {10.7, -2.5, 1};

**float** p1[] = {10.7, -9, 3}, s1[] = {12.7, -9, 3}, q1[] = {10.7, -1, 3}, r1[] = {12.7, -1, 3};

**float** p2[] = {0.5, -1, 1}, s2[] = {5.5, -1, 1}, q2[] = {0.5, -9, 1}, r2[] = {5.5, -9, 1};

**float** p3[] = {-15, -6.5, 1}, q3[] = {-15, -3.5, 1}, r3[] = {0.5, -2.5, 1}, s3[] = {0.5, -7.5, 1};

**float** p4[] = {-13, -6.5, 1}, q4[] = {-13, -6.5, 2.5}, r4[] = {0.5, -7.5, 3.5}, s4[] = {0.5, -7.5, 1};

**float** p5[] = {-13, -3.5, 1}, q5[] = {-13, -3.5, 2.5}, r5[] = {0.5, -2.5, 3.5}, s5[] = {0.5, -2.5, 1};

**float** p6[] = {5.5, -2.5, 1}, q6[] = {5.5, -2.5, 3.5}, r6[] = {10.7, -2.5, 3.5}, s6[] = {10.7, -2.5, 1};

**float** p7[] = {5.5, -7.5, 1}, q7[] = {5.5, -7.5, 3.5}, r7[] = {10.7, -7.5, 3.5}, s7[] = {10.7, -7.5, 1};

**float** p8[] = {5.5, -7.5, 3.5}, q8[] = {10.7, -7.5, 3.5}, r8[] = {10.7, -6, 3.5}, s8[] = {5.5, -6, 3.5};

**float** p9[] = {5.5, -2.5, 3.5}, q9[] = {5.5, -4, 3.5}, r9[] = {10.7, -4, 3.5}, s9[] = {10.7, -2.5, 3.5};

**float** p10[] = {5.5, -4, 3.5}, q10[] = {10.7, -4, 3.5}, r10[] = {10.7, -5, 4.5}, s10[] = {5.5, -5, 5.5};

**float** p11[] = {5.5, -6, 3.5}, q11[] = {10.7, -6, 3.5}, r11[] = {10.7, -5, 4.5}, s11[] = {5.5, -5, 5.5};

**float** p12[] = {10.7, -9, 2}, q12[] = {10.7, -9, 4}, r12[] = {12.7, -9, 4}, s12[] = {12.7, -9, 2};

**float** p13[] = {10.7, -1, 2}, q13[] = {10.7, -1, 4}, r13[] = {12.7, -1, 4}, s13[] = {12.7, -1, 2};

**float** p14[] = {0.5, -1, 1}, q14[] = {0.5, -1, 3}, r14[] = {5.5, -1, 3}, s14[] = {5.5, -1, 1};

**float** p15[] = {0.5, -9, 1}, q15[] = {0.5, -9, 3}, r15[] = {5.5, -9, 3}, s15[] = {5.5, -9, 1};

**float** p16[] = {0.5, -1, 1}, q16[] = {0.5, -1, 3}, r16[] = {0.5, -2.5, 3.5}, s16[] = {0.5, -2.5, 1};

**float** p17[] = {0.5, -7.5, 1}, q17[] = {0.5, -7.5, 3.5}, r17[] = {0.5, -9, 3}, s17[] = {0.5, -9, 1};

**float** p18[] = {5.5, -1, 1}, q18[] = {5.5, -1, 3}, r18[] = {5.5, -2.5, 3.5}, s18[] = {5.5, -2.5, 1};

**float** p19[] = {5.5, -7.5, 1}, q19[] = {5.5, -7.5, 3.5}, r19[] = {5.5, -9, 3}, s19[] = {5.5, -9, 1};

**float** p20[] = {10.7, -7.5, 1}, q20[] = {10.7, -7.5, 3.5}, r20[] = {10.7, -2.5, 3.5},

s20[] = {10.7, -2.5, 1};

**float** p21[] = {4, -2.5, 3.5}, q21[] = {5.5, -2.5, 3.5}, r21[] = {5.5, -7.5, 3.5}, s21[] = {4, -7.5, 3.5};

**enum** { *// Constants for different views*

***HELICOPTER***, ***FRONT***, ***SIDE***, ***BACK***

} viewpoint = ***FRONT***;

**int** MID = 570; *//Distance of the car on the track from the centre of the track*

**int** start = 0;

**char** KEY; *//Variable that stores key pressed by user*

**float** angle; *//Rotation angle for car*

**float** carx = 0, cary = 570; *//Variables that specify position of the car*

**int** rot = 0; *//rotation angle for the wheels*

*//Function to generate a cone*

**void** cone() {

**float** i, x, y, r = 10;

glColor3f(0.0, 0.7, 0.2);

glBegin(**GL\_TRIANGLE\_FAN**);

glVertex3f(0, 0, 20);

**for** (i = 0; i <= 361; i += 2) {

x = r \* cos(i \* **c**);

y = r \* sin(i \* **c**);

glVertex3f(x, y, 0);

}

glEnd();

}

*//Fuction to draw the track*

**void** track(**float** R1, **float** R2) {

**float** X, Y, Z;

**int** y;

glBegin(**GL\_QUAD\_STRIP**);

**for** (y = 0; y <= 361; y += 1) {

X = R1 \* cos(**c** \* y);

Y = R1 \* sin(**c** \* y);

Z = -1;

glVertex3f(X, Y, Z);

X = R2 \* cos(**c** \* y);

Y = R2 \* sin(**c** \* y);

Z = -1;

glVertex3f(X, Y, Z);

}

glEnd();

}

*//Function that generates a cylinder*

**void** cylinder(**float** r, **float** l) {

**float** x, y, z;

**int** d;

glBegin(**GL\_QUAD\_STRIP**);

**for** (d = 0; d <= 362; d += 1) {

x = r \* cos(**c** \* d);

z = r \* sin(**c** \* d);

y = 0;

glVertex3f(x, y, z);

y = l;

glVertex3f(x, y, z);

}

glEnd();

}

*//Function that generates tree with cone shaped tree top*

**void** tree(**float** a, **float** b) { *//Tree trunk*

glColor3f(0.9, 0.3, 0);

glPushMatrix();

glTranslatef(a, b, -1);

glRotatef(90, 1, 0, 0);

cylinder(3, 15);

glPopMatrix();

*//Cone shaped tree top*

glPushMatrix();

glTranslatef(a, b, 8);

cone();

glPopMatrix();

}

*//Functin that generates tree with sphere shaped tree top*

**void** tree2(**float** a, **float** b) {

*//Tree trunk*

glColor3f(1, 0.2, 0);

glPushMatrix();

glTranslatef(a, b, -1);

glRotatef(90, 1, 0, 0);

cylinder(6, 25);

glPopMatrix();

*//Sphere shaped tree top*

glColor3f(0, 1, 0.3);

glPushMatrix();

glTranslatef(a, b, 45);

glutSolidSphere(30, 10, 10);

glPopMatrix();

}

*//Function to generate the sides of the tyres*

**void** alloy(**float** R1, **float** R2) {

**float** X, Y, Z;

**int** y;

glColor3f(0, 0, 0);

glBegin(**GL\_QUAD\_STRIP**);

**for** (y = 0; y <= 361; y += 1) {

X = R1 \* cos(**c** \* y);

Z = R1 \* sin(**c** \* y);

Y = 0;

glVertex3f(X, Y, Z);

X = R2 \* cos(**c** \* y);

Z = R2 \* sin(**c** \* y);

Y = 0;

glVertex3f(X, Y, Z);

}

glEnd();

}

*//Function to draw the spokes of the wheel*

**void** actall(**float** R1, **float** R2) {

**float** X, Y, Z;

**int** i;

glBegin(**GL\_QUADS**);

**for** (i = 0; i <= 361; i += 120) {

glColor3f(0, 0.5, 0.5);

X = R1 \* cos(**c** \* i);

Y = 0;

Z = R1 \* sin(**c** \* i);

glVertex3f(X, Y, Z);

X = R1 \* cos(**c** \* (i + 30));

Y = 0;

Z = R1 \* sin(**c** \* (i + 30));

glVertex3f(X, Y, Z);

X = R2 \* cos(**c** \* (i + 30));

Y = 0;

Z = R2 \* sin(**c** \* (i + 30));

glVertex3f(X, Y, Z);

X = R2 \* cos(**c** \* i);

Y = 0;

Z = R2 \* sin(**c** \* i);

glVertex3f(X, Y, Z);

}

glEnd();

}

*//Function to draw a circle*

**void** circle(**float** R) {

**float** X, Y, Z;

**int** i;

glBegin(**GL\_POLYGON**);

**for** (i = 0; i <= 360; i++) {

X = R \* cos(**c** \* i);

Z = R \* sin(**c** \* i);

Y = 0;

glVertex3f(X, Y, Z);

}

glEnd();

}

*//Function to draw a quadrilateral*

**void** rect(**float** p[], **float** q[], **float** r[], **float** s[]) {

glBegin(**GL\_POLYGON**);

glVertex3fv(p);

glVertex3fv(q);

glVertex3fv(r);

glVertex3fv(s);

glEnd();

}

*//Function to generate car driver*

**void** driver() {

glColor3f(0.5, 0.2, 0.8);

*//Legs*

glPushMatrix();

glTranslatef(3, -3.5, 1.5);

glRotatef(90, 0, 0, 1);

cylinder(0.4, 3);

glPopMatrix();

glPushMatrix();

glTranslatef(3, -6.5, 1.5);

glRotatef(90, 0, 0, 1);

cylinder(0.4, 3);

glPopMatrix();

*//Hands*

glPushMatrix();

glTranslatef(3, -3.5, 2.5);

glRotatef(90, 0, 0, 1);

cylinder(0.4, 3);

glPopMatrix();

glPushMatrix();

glTranslatef(3, -6.5, 2.5);

glRotatef(90, 0, 0, 1);

cylinder(0.4, 3);

glPopMatrix();

*//Head*

glPushMatrix();

glTranslatef(3, -5, 4);

glutSolidSphere(1.0, 20, 16);

glPopMatrix();

*//Body*

glPushMatrix();

glTranslatef(3, -5, 1);

glRotatef(90, 1, 0, 0);

cylinder(1, 2);

glPopMatrix();

*//Circle*

glPushMatrix();

glTranslatef(3, -5, 3);

glRotatef(90, 1, 0, 0);

circle(1);

glPopMatrix();

}

*//Function generating scenery using functions track( ),tree( ),tree2( )*

**void** scenery() {

**float** x, y;

**int** p;

*//Background*

glColor3f(0.0, 0.0, 0.0);

glPushMatrix();

glRotatef(90, 1, 0, 0);

cylinder(1000, 1000);

glPopMatrix();

*//Ground*

glColor3f(0, 1, 0);

glPushMatrix();

glTranslatef(0, 0, -1.1);

glRotatef(90, 1, 0, 0);

circle(1100);

glPopMatrix();

*//Track*

glColor3f(0.3, 0.3, 0.6);

track(600, 540);

*//Cone shaped trees*

**for** (p = 0; p <= 360; p += 30) {

x = 700 \* cos(**c** \* p);

y = 700 \* sin(**c** \* p);

tree(x, y);

}

*//Sphere shaped trees*

**for** (p = 100; p <= 460; p += 30) {

x = 800 \* cos(**c** \* p);

y = 800 \* sin(**c** \* p);

tree2(x, y);

}

}

*//Function to draw triangles*

**void** tri(**float** a[], **float** b[], **float** z[]) {

glBegin(**GL\_TRIANGLES**);

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(z);

glEnd();

}

*//Function that has calls to other functions to generate wheels along with axle*

**void** wheels() {

*//axle*

glColor3f(0, 0.5, 0.3);

cylinder(0.4, 9);

*//1st Wheel*

glColor3f(0, 0, 0);

cylinder(2, 2);

alloy(2, 1.4);

actall(1.4, 0.8);

glColor3f(0, 0.5, 0.4);

circle(0.8);

glPushMatrix();

glTranslatef(0, 2, 0);

alloy(2, 1.4);

actall(1.4, 0.8);

glColor3f(0, 0.5, 0.4);

circle(0.8);

glPopMatrix();

*//2nd Wheel*

glPushMatrix();

glTranslatef(0, 8, 0);

glColor3f(0, 0, 0);

cylinder(2, 2);

alloy(2, 1.4);

actall(1.4, 0.8);

glColor3f(0, 0.5, 0.4);

circle(0.8);

glPopMatrix();

glPushMatrix();

glTranslatef(0, 10, 0);

actall(1.4, 0.8);

alloy(2, 1.4);

glColor3f(0, 0.5, 0.4);

circle(0.8);

glPopMatrix();

}

*//Function that generates the chassis of the car*

**void** chassis() {

*//Parameters For glMaterialfv() function*

GLfloat specular[] = {0.7, 0.7, 0.7, 1.0};

GLfloat ambient[] = {1, 1, 1, 1}, diffuse[] = {0.7, 0.7, 0.7, 1};

GLfloat full\_shininess[] = {100.0};

*//Material Properties*

glMaterialfv(**GL\_FRONT**, **GL\_AMBIENT**, ambient);

glMaterialfv(**GL\_FRONT**, **GL\_SPECULAR**, specular);

glMaterialfv(**GL\_FRONT**, **GL\_DIFFUSE**, diffuse);

glMaterialfv(**GL\_FRONT**, **GL\_SHININESS**, full\_shininess);

glColor3f(0, 0.2, 0.9);

rect(p, q, r, s);

rect(p2, q2, r2, s2);

rect(p3, q3, r3, s3);

rect(p4, q4, r4, s4);

rect(p5, q5, r5, s5);

rect(q5, q4, r4, r5);

rect(p6, q6, r6, s6);

rect(p7, q7, r7, s7);

rect(p8, q8, r8, s8);

rect(p9, q9, r9, s9);

glColor3f(1, 0.6, 0);

rect(p1, q1, r1, s1);

rect(q5, q4, p3, q3);

tri(p4, q4, p3);

tri(p5, q5, q3);

rect(p10, q10, r10, s10);

rect(p11, q11, r11, s11);

rect(r16, r18, q18, q16);

rect(q17, q19, r19, r17);

rect(p21, q21, r21, s21);

glColor3f(0, 0.2, 0.9);

rect(p12, q12, r12, s12);

rect(p13, q13, r13, s13);

rect(p14, q14, r14, s14);

rect(p15, q15, r15, s15);

rect(p16, q16, r16, s16);

rect(p17, q17, r17, s17);

rect(p18, q18, r18, s18);

rect(p19, q19, r19, s19);

rect(r18, q19, p19, s18);

rect(p20, q20, r20, s20);

}

*//Function that that has function calls to chassis(),tyrea(),*

*//tyreb(),driver() to generate the car with wheels rotating*

**void** car() {

glPushMatrix();

glRotatef(180, 0, 0, 1);

chassis();

glPushMatrix();

glTranslatef(8, -10, 1);

glRotatef(rot, 0, 1, 0);

wheels();

glPopMatrix();

glPushMatrix();

glTranslatef(-12, -10, 1);

glRotatef(rot, 0, 1, 0);

wheels();

glPopMatrix();

driver();

rot += 90;

**if** (rot > 360) rot -= 360;

glPopMatrix();

}

*//Keyboard Callback Function*

**void** keys(**unsigned char** key, **int** x, **int** y) {

KEY = key;

**if** (key == **'E'** || key == **'e'**) { start = 0; }

**if** (key == **'G'** || key == **'g'**) { start = 1; }

}

*//Function that generates a particular view of scene depending on view selected by //user*

**void** view() {

**float** pos[] = {1000, 1000, 2000, 1};*//Position of the light source*

**switch** (viewpoint) {

**case *FRONT***:

gluLookAt(15.0, 5.0, 20, 0.0, 0.0, 4.0, 0.0, 0.0, 1.0);

car();

glPushMatrix();

glRotatef(angle \* **RAD\_TO\_DEG**, 0.0, 0.0, 1.0);

glTranslatef(-carx, -cary, 0);

glLightfv(**GL\_LIGHT0**, **GL\_POSITION**, pos);

scenery();

glPopMatrix();

**break**;

**case *BACK***:

gluLookAt(-12.0, 6.0, 13, 15.0, 6.0, 2.0, 0.0, 0.0, 1.0);

car();

glPushMatrix();

glRotatef(**RAD\_TO\_DEG** \* angle, 0.0, 0.0, 1.0);

glTranslatef(-carx, -cary, 0);

glLightfv(**GL\_LIGHT0**, **GL\_POSITION**, pos);

scenery();

glPopMatrix();

**break**;

}

}

*//Idle Callback Function*

**void** idle() {

**if** (start == 1) {

angle += 0.05;

**if** (angle == **TWO\_PI**) {

angle -= **TWO\_PI**;

}

carx = MID \* sin(angle);

cary = MID \* cos(angle);

**switch** (KEY) {

**case 'F'**:

**case 'f'**:

viewpoint = ***FRONT***;

**break**;

**case 'B'**:

**case 'b'**:

viewpoint = ***BACK***;

**break**;

}

glutPostRedisplay();

}

}

**void** init() {

GLfloat amb[] = {1, 1, 1, 1}, diff[] = {1, 1, 1, 1}, spec[] = {1, 1, 1, 1};

glLoadIdentity();

glLightfv(**GL\_LIGHT0**, **GL\_AMBIENT**, amb);

glLightfv(**GL\_LIGHT0**, **GL\_DIFFUSE**, diff);

glLightfv(**GL\_LIGHT0**, **GL\_SPECULAR**, spec);

glLightModeli(**GL\_LIGHT\_MODEL\_TWO\_SIDE**, **GL\_TRUE**);

glEnable(**GL\_COLOR\_MATERIAL**);

glEnable(**GL\_LIGHTING**);

glEnable(**GL\_LIGHT0**);

glEnable(**GL\_DEPTH\_TEST**);

glClearColor(1, 1, 1, 1);

}

*//Display Callback Function*

**void** display() {

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

glMatrixMode(**GL\_MODELVIEW**);

glLoadIdentity();

view();

glutSwapBuffers();

}

*//Reshape Function*

**void** reshape(**int** w, **int** h) {

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

glMatrixMode(**GL\_PROJECTION**);

glLoadIdentity();

gluPerspective(100, (GLfloat) w / (GLfloat) h, 1, 2000.0);

glMatrixMode(**GL\_MODELVIEW**);

glLoadIdentity();

}

*//Main Function*

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

glutInit(&argc, argv);

glutInitDisplayMode(**GLUT\_DOUBLE** | **GLUT\_RGB** | **GLUT\_DEPTH**);

glutInitWindowPosition(500, 500);

*//FGAPI void FGAPIENTRY glutInitWindowSize( int width, int height );*

glutInitWindowSize(500, 500);

glutCreateWindow(**"Car G-start E-stop B-Back F-Front"**);

glutDisplayFunc(display);

glutIdleFunc(idle);

glutKeyboardFunc(keys);

glutReshapeFunc(reshape);

init();

glutMainLoop();

**return** 0;

}



