IUBAT-International University of Business Agriculture and Technology

Course Name:computer Graphics
Course Code:csc 455

Submitted BY

Name	ID	Section
Sarmin Akter	20103070	D
Farjana Rahman	20103201	D
Sabrina Sultana Sweety	20103202	A

Submitted TO
Mr Krishna Das
Assistant Professor
Department of CSE at IUBAT

Abstract:

The "A Moving Car Scenario" is a simple interactive animation program implemented using OpenGL and GLUT (OpenGL Utility Toolkit). The program simulates a scene with a main car, a second car, and a third car moving on a road surrounded by hills. The main car can be controlled by the user using the keyboard keys 'a' (left) and 'd' (right). The second car moves back and forth on the road, and the third car moves horizontally from right to left. The program aims to provide an engaging and interactive visual experience to the user.

CONTENTS

1.Introduction:

The "A Moving Car Scenario" code is a simple interactive graphics program written in C utilizing OpenGL and GLUT (OpenGL Utility Toolkit) libraries. The program simulates a visually engaging environment where cars move on a road against the backdrop of hills and a setting sun. The user can interact with the main car, controlling its movement horizontally on the screen using the 'a' and 'd' keys.

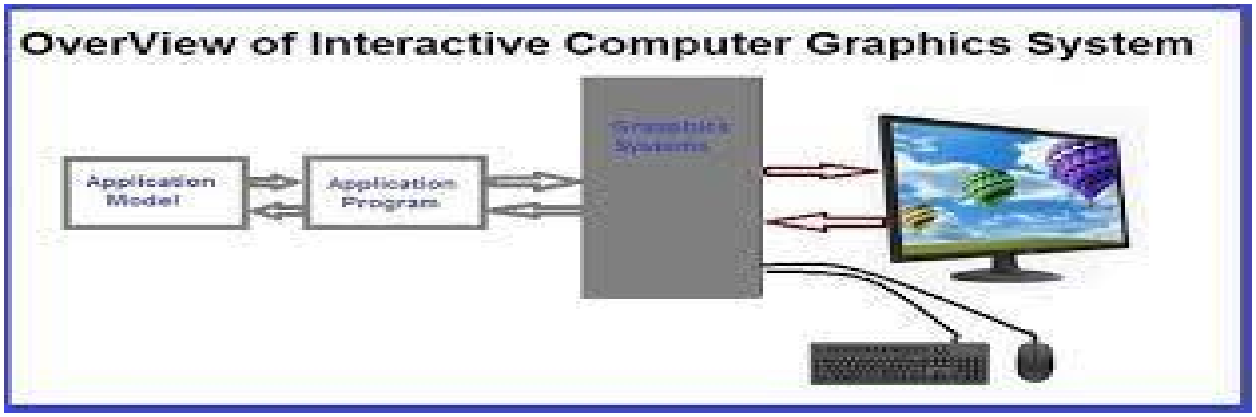
The scene is set on a two-lane road, and the main car, represented by a red polygon, moves horizontally with animated wheels. The road is visually distinguished with different colors for better visualization. The background includes hills, creatively drawn using polygons, to provide an aesthetic landscape effect. A stylized sun is also featured in the scene.

The two cars, represented by blue and green polygons, move autonomously along the road. The second car's lane can be switched using the spacebar, and its movement adds an element of interaction between the main car and other two cars.

2.LITERATURE SURVEY

CG (Computer graphics) started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computers themselves. It includes the creation, storage, and manipulation of models and images of objects. These models include physical, mathematical, engineering, architectural, and even conceptual or abstract structures, natural phenomena, and so on.
Interactive Graphics: This is an interactive graphics.

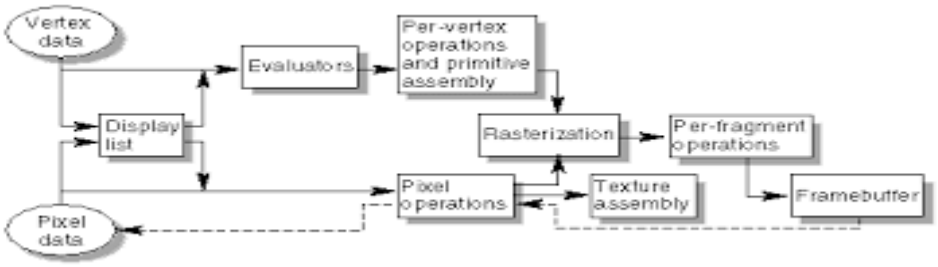
2.1 Interactive Graphics. Interactive graphics refers to the dynamic and real-time manipulation of computer-generated visual content through user input. It involves creating digital graphics that respond to user actions, enabling users to interact with and modify the displayed images or scenes in immediate and intuitive ways. This interaction can occur in various domains, including video games, simulations, virtual environments, user interfaces, and more.



2.2 About open-GL

Interactive graphics refers to the dynamic and real-time manipulation of computer-generated visual content through user input. It involves creating digital graphics that respond to user actions, enabling users to interact with and modify the displayed images or scenes in immediate and intuitive ways. This interaction can occur

in various domains, including video games, simulations, virtual environments, user interfaces, and more.



2.3 Advantages of OpenGL-

- With different 3D accelerators, by presenting the programmer to hide the complexities of interfacing with a single, uniform API.

- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).

3. REQUIREMENT SPECIFICATION

3.1 Hardware requirements- Dual core processor, 2 gb or more
RAM, a standard keyboard, compatible mouse and a VGA monitor. If the user wants to save the created files a secondary storage medium can be used.

3.2 Software requirements- The graphics package has been designed for windows OS, hence the machine must have Eclipse installed preferably 7.02.83 or later versions with the glut library functions installed. Other graphics libraries are used.

3.3 Development platform- Ubuntu 11.03 and Eclipse

3.4 Language used in coding- C++ language.

3.5 Functional Requirements-

void circle(): This function is use to make the round shape of the object on the monitor.
Void sun(): This function is for making sun on the sky by horizontal and vertical.
Void hills(): To draw on the screen triangle shape .
Void car(): This function used to draw the object bus in the scene.

4.1 Pseudo Code:

The main algorithm for the Car-Movement can be written as below:
Initially define the void circle () function and keep it running throughout the course of the program.
Initialize the void sun (), void car () and void hills () functions.
Initialize the keyboard and mouse interface functions.
Keep the above scenery in place on the window and wait for the user to press the start animation button.
As soon as the above step is performed, the following actions should be performed simultaneously :
The first car should move continuously. The second car should be able to change its lane when it's necessary by using keyboard Instruction. The hills, sun, car can change their color through the given instruction. The size of car windows, wheels, sun, hill etc. can be expanded through instruction. During any time of the execution of the program, if the user presses the stop animation button, the execution should halt (pause) and resume back if start animation is again pressed. During the entire course of execution, if the user presses the exit button, the window should exit immediately.

4.2 Analysis:

The functions and their algorithms are as follows:
Void sun (): Since we need the sun we‘ve use the combination of red (255) and green(215). The glColor3ub() function sets the current color to a shade of yellow (RGB: 255, 215, 0), which is typically associated with the color of the sun. The sun() function is called with parameters (20, 20, 175, 450). It appears that sun() is expected to draw a circular shape, possibly representing the sun, at the coordinates (175, 450) with a radius of 20.
Void hills(): This function represents the hill structure in a combination of RGB where (184, 134,11) represents a shade of brownish color. glBegin(GL_POLYGON) This indicates the start of a polygon drawing operation. The parameter GL_POLYGON specifies that a polygon will be drawn using the subsequent glVertex calls.

Void car(): We have translated a two car over the road. Constructing the second car was again a challenge but was easier to implement once we had got the first car done. It was implemented by drawing a set of regular polygons and then merging them in

parts to look like a car. We used 3 sets each consisting of 4 points each to get the layout followed by a set of 16 points for the
carrier.

Void road():
This section draws a green rectangle representing the ground in both side. Between them there is a black rectangle .
Void display():
This function basically displays all the above described functions on the screen as we flush the output onto the screen form the frame buffer.

Void init() and Void main():
These are the typical functions which appear in almost all programs and are described in chapter3 in detail.

Void keyboard():
This function basically changes the direction of cars. The colors that the car can have are red, green, blue, yellow, cyan and magenta. The main concept behind the usage of the above 6 colors are that they are the additive and the subtractive colors:
Additive color:
Form a color by adding amounts of three primarie CRTs, projection systems, positive film Primaries are Red (R), Green (G), Blue (B)

Subtractive color:
Form a color by filtering white light with cyan (C), Magenta (M), and Yellow (Y) filters Light-material interactions Printing Negative film

Fig: 4.1

The above figure of the eye shows the Rhodes and cones. Rhodes is responsible to detect the additive colors and the cones are in charge of detecting the subtractive colors.

Void main ():
This function puts the whole program together. It says which function to execute first and which one at the end. However, here we have used int main () since eclipse expects the main to have a return value.

IMPLEMENTATION

The Car movement can be implemented using some of the OpenGL inbuilt functions along with some user defined functions. The inbuilt OpenGL functions that are used mentioned under the FUNCTIONS USED category. The user defined functions are mentioned under USER DEFINED FUNCTIONS category.

4.1 Functions Used –
Void glColor3ub (float red, float green, float blue):

This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. The ‘f‘ gives the data type float. The arguments are in the order RGB (Red, Green and Blue). The color of the pixel can be specified as the combination of these 3 primary colors.

Void glClearColor(int red, int green, int blue, int alpha):
This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

Void KeyboardFunc():
Where func () is the new keyboard callback function. KeyboardFunc 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. The key callback
parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window for relative coordinates when the key gets pressed. Prototype is as given below:
void keyboardFunc(unsigned char key, int x, int y);
When a new window is created, no keyboard callback is initially registered, and the ASCII keystrokes that are within the output window are ignored. Passing NULL to KeyboardFunc disables the generation of keyboard callbacks.
Void glFlush():
Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. GLflush () empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Void glMatrixMode(GL_PROJECTION):

Where —model‖ specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted are:
GL_MODELVIEW, GL_PROJECTION and GL_TEXTURE
The initial value is GL_MODELVIEW.
The function GlMatrixMode sets the current matrix mode. Mode can assume one of these values:
GL_MODELVIEW : Applies matrix operations to the model view matrix stack.
GL_PROJECTION: Applies matrix operations to the projection matrix stack.

Void glutInit(&argc, argv):
GlutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options. GlutInit also processes command line options, but the specific options parse are window system dependent.
Void glutReshapeFunc (void (*func) (int width, int height)):
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.
If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply
glOrtho ( ):

Syntax: void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
The function defines an orthographic viewing volume with all parameters measured from the center of the projection plane.
void 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 stop.

glutPostRedisplay()
GlutPostRedisplay, glutPostWindowRedisplay — marks the current or specified window as needing to be redisplayed.

4.2 USER DEFINED FUNCTIONS:

void sun():
This function depicts the sun by drawing circle on the window. This function makes use of the OpenGL functions to define the window size, radius of the circle , to provide the color for sun and make it to translate in required direction.

Void hill():
This function depicts the hill in the scene. This function designs the hill by Using polygon. This function defines hills by making use of OpenGL functions.

Void car():
This function used to draw the object car in the scene. A car would be created by plotting the points at the proper distances to resemble the shape of a car and then these points would be joined with the lines to make the car like image complete.

Void road():
This function used to draw the road for moving the cars. There is three lane. These are drawn using the OpenGL inbuilt functions.

void keyboardFunc(unsigned char key, int x, int y):
This function used to provide the keyboard interface to the user. Using this function we can change the direction of the car pressing the corresponding keys. For example if we press the key 'a' move the main car to the left, similarly if we press the button 'd' move the main car to the right if we press ’ ‘ switch lanes for the second car
Void display(void):
In this function first we should print the instructions that we would be displayed on the pop-up window.
Int main(int argc, char** argv):
Here we call all the function we defined previously in the program and this function creates an output window.

DISCUSSION AND SHAPSHOTS

6.1 Discussions

Before we start the project, there we have many other particle topics that we are thinking to design using the OpenGL functions.For example you'll need to utilize several key features in our interactive car model for the primary level students for learning while playing.
Vertex and Fragment Shaders: These shaders handle the rendering of vertices and fragments (pixels) on the screen. You'll use vertex shaders to transform the car's vertices and fragment shaders to color and shade the pixels.

Model-View-Projection (MVP) Matrix: The MVP matrix combines model, view, and projection matrices to position and project your car's model onto the screen.

Buffers: You'll need vertex buffers to store the car's geometry and attributes, such as positions, normals, and texture coordinates.Texture Mapping: Apply textures to the car's geometry to give it a realistic appearance. This involves loading texture images and mapping them onto the appropriate parts of the car.

Uniforms: Use uniforms to send data from your application to the shaders. This can include transformation matrices for MVP, lighting parameters, and other variables that influence rendering.

Input Handling: Implement a way to handle user input, such as keyboard controls or mouse interactions, to move the car.

Animation: Utilize interpolation techniques to smoothly animate the car's movement, rotation, and other transformations over time.

Depth Testing: Enable depth testing to ensure that objects appear correctly in relation to their positions in 3D space.

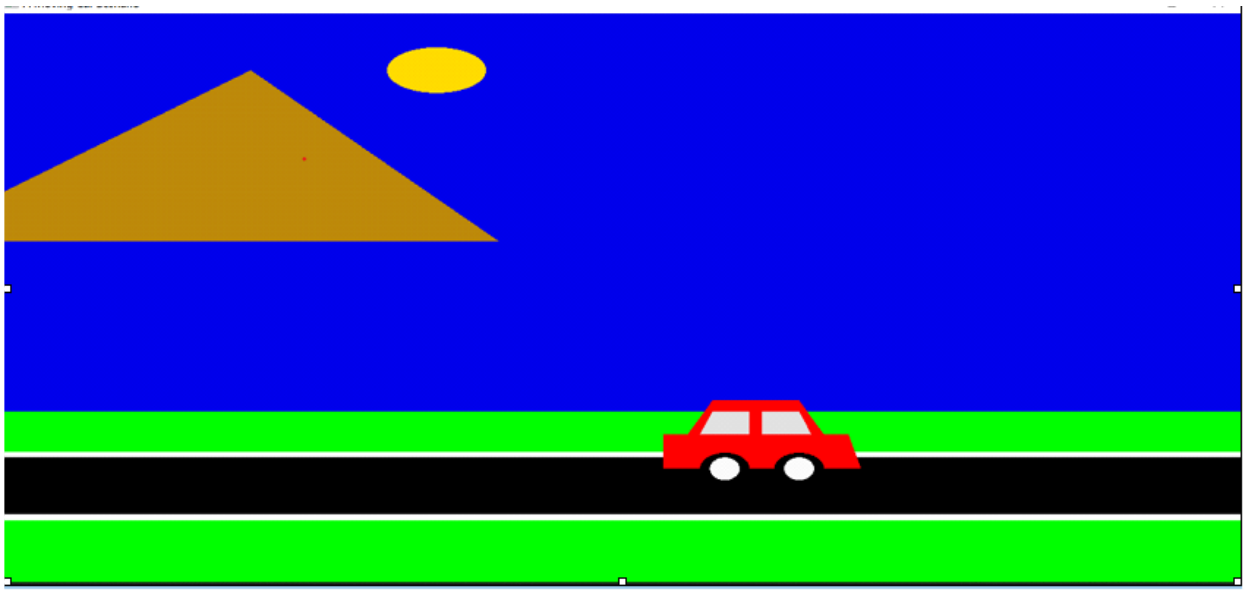
Lighting: Apply lighting models to make the car's surfaces react to light sources realistically. This involves calculating light intensities, specular highlights, and shadowing.

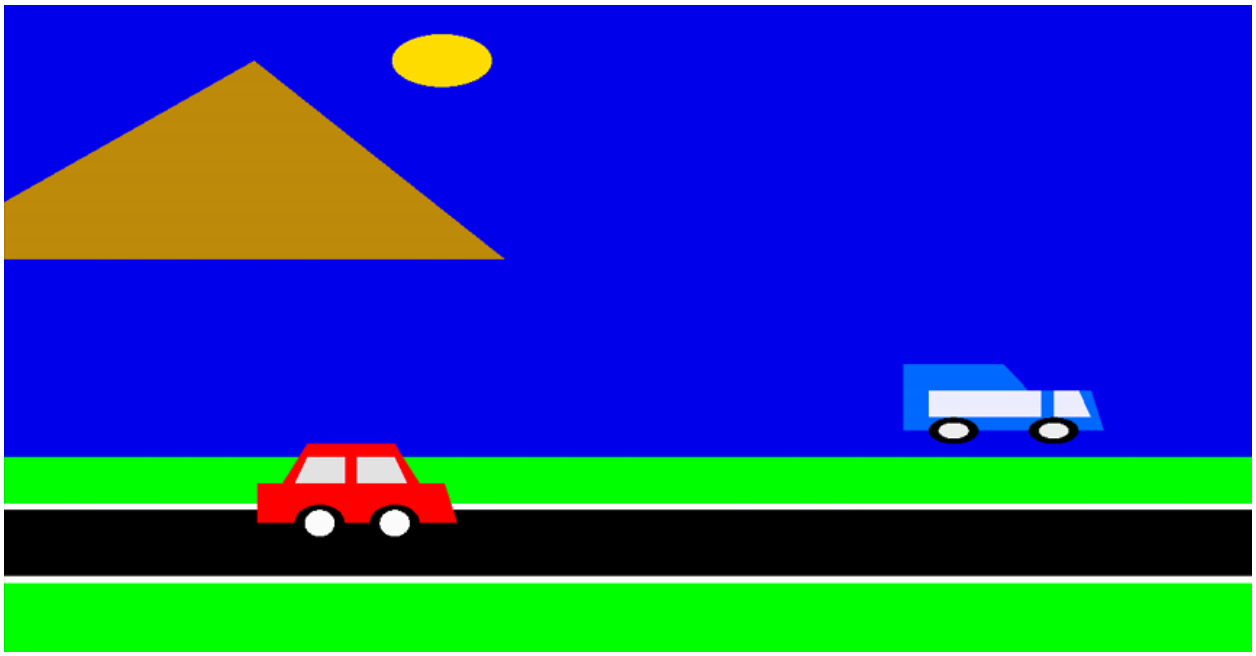
Camera Control: Set up a camera that can be positioned and oriented to give different views of the scene. This allows users to follow the moving car from different angles.Here creating a moving car involves a combination of geometry, shaders, matrices, and user interaction. Start with simple movements and gradually add complexity as you become more comfortable with the OpenGL pipeline.

For designing the car model it involves creating a 3D digital representation of a vehicle, refining itsappearance and details, simulating lighting and other effects, and finally rendering high-quality images
or animations. This process allows designers to explore and refine car designs before moving on to physical production.We can control its speed as well as its moving lane as there we have 2 lanes in our Animation also. we can change its wheels colors and move the car backwards also.By the movement we are going to make understand student that if we increase the speed the car can pass the road very fast
again with the low speed the car need more time to reach the ending points
For designing the hill we use a Triangle near the road. We can change the color of it and change its position according to its perspective of viewing point.creating a hill in graphic animated design involves sculpting the terrain, generating a 3D mesh, applying textures and details, setting up lighting, and using rendering techniques to achieve a realistic and visually appealing representation. The process is used to seamlessly blend the hill into the larger design and create immersive environments.

Our next part is making the sun. We use a circle to define a sun in the window. Wecan make it change by applying different color values. Begin by drawing a simple circle to represent the main body of the sun.The circle serves as the foundation for the sun's design.Extend lines outward from the center dot to the edge of the circle. These lines represent the sun's radiation and can be drawn as straight lines or slightly curved rays.Apply a radial color gradient to the sun's circle, with the center being the brightest point and the outer edges transitioning to a softer, warmer hue.
The most important part was making the road lane. There we have 2 lanes in our project.For perspective the second lane is not fully viewable. Begin by drawing a simple straight line or a gently curving path to outline the road's basic shape. This will serve as the foundation for the road design.Determine the width of the road based on the scale and perspective of your design. In perspective, the road might appear narrower as it recedes into the distance.

6.2 SNAPSHOTS





Conclusion and Future scope:

The "A Moving Car Scenario" code successfully demonstrates a basic interactive graphics application using OpenGL and GLUT. The program presents an engaging and visually appealing scene where cars move on a road against a backdrop of hills and a setting sun. The main car is controlled by the user, allowing them to move it horizontally on the screen using the 'a' and 'd' keys. The AI-controlled second car moves back and forth horizontally on the road, and the user can switch its lane using the spacebar. Additionally, a third AI-controlled car moves horizontally from left to right. features and realistic physics to create more sophisticated and entertaining visual simulations in the future.

BIBLIOGRAPHY

8.1 Web References:

https://www.opengl.org/
https://www.google.com/
https://sourcecode.com/
www.wikipedia.org

APPENDICES:

Source Code:

```
#include <GL/gl.h>
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>

float bx = 10;      // x-coordinate of the main car
float cx2 = 50;    // Initial x-coordinate of the second car
float dx2 = -1.5;   // Movement speed of the second car
float cx3 = 300;    // Initial x-coordinate of the third car
float dx3 = 2.0;    // Movement speed of the third car
int secondCarLane = 0;  // 0 for right lane, 1 for left lane

void circle(GLfloat rx, GLfloat ry, GLfloat cx, GLfloat cy)
{
   glBegin(GL_POLYGON);
   glVertex2f(cx, cy);
   for (int i = 0; i <= 360; i++)
   {
      float angle = i * 3.1416 / 180;
      float x = rx * cos(angle);
      float y = ry * sin(angle);
      glVertex2f((x + cx), (y + cy));
   }
   glEnd();
}

void sun(GLfloat rx, GLfloat ry, GLfloat cx, GLfloat cy)
{
   glBegin(GL_POLYGON);
```

```
    glVertex2f(cx, cy);
    for (int i = 0; i <= 360; i++)
    {
        float angle = i * 3.1416 / 180;
        float x = rx * cos(angle);
        float y = ry * sin(angle);
        glVertex2f((x + cx), (y + cy));
    }
    glEnd();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.9, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 500, 0.0, 500); // window size
}

void hills() {
    glColor3ub(184, 134, 11);
    glBegin(GL_POLYGON);
    glVertex2d(-40, 300);
    glVertex2d(200, 300);
    glVertex2d(100, 450);
    glEnd();
    // ... Existing hills function code ...
}

void drawSecondCar() {
    glPushMatrix();
    glTranslatef(cx2, secondCarLane * 70, 0);

    // Draw the second car here
    glColor3ub(0, 102, 255); // Blue color
    glBegin(GL_POLYGON);
    glVertex2d(410, 100);
    glVertex2d(490, 100);
    glVertex2d(485, 130);
    glVertex2d(460, 130);
    glVertex2d(450, 150);
```

```
    glVertex2d(410, 150);
    glEnd();

    // Car windows
    glColor3ub(230, 230, 255); // Light blue color
    glBegin(GL_POLYGON);
    glVertex2d(420, 110);
    glVertex2d(465, 110);
    glVertex2d(465, 130);
    glVertex2d(420, 130);
    glEnd();

    glBegin(GL_POLYGON);
    glVertex2d(470, 110);
    glVertex2d(485, 110);
    glVertex2d(480, 130);
    glVertex2d(470, 130);
    glEnd();

    // Car wheels
    glColor3ub(0, 0, 0); // Black color
    circle(10, 10, 430, 100);
    circle(10, 10, 470, 100);

    glColor3ub(230, 230, 230); // Light gray color
    circle(6, 6, 430, 100);
    circle(6, 6, 470, 100);

    glPopMatrix();
}

void drawThirdCar() {
    glPushMatrix();
    glTranslatef(cx3, 0, 0);

    // Draw the third car here, similar to the first car code in the display() function.
    glColor3ub(0, 255, 0);
    glBegin(GL_POLYGON);
    glVertex2d(410, 200);
    glVertex2d(490, 200);
```

```
    glVertex2d(485, 230);
    glVertex2d(410, 230);
    glEnd();

    // ... Add more code to draw the rest of the third car ...

    glPopMatrix();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    //Ground Color
    glColor3ub(0, 255, 0);
    glBegin(GL_POLYGON);
    glVertex2d(0, 0);
    glVertex2d(500, 0);
    glVertex2d(500, 150);
    glVertex2d(0, 150);
    glEnd();

    // road

    glColor3ub(255, 255, 255);
    glBegin(GL_POLYGON);
    glVertex2d(0, 55);
    glVertex2d(500, 55);
    glVertex2d(500, 115);
    glVertex2d(0, 115);
    glEnd();

    glColor3ub(0, 0, 0);
    glBegin(GL_POLYGON);
    glVertex2d(0, 60);
    glVertex2d(500, 60);
    glVertex2d(500, 110);
    glVertex2d(0, 110);
    glEnd();
```

```
// hills
hills();

// sun design
glColor3ub(255, 215, 0);
sun(20, 20, 175, 450);

glPushMatrix();
glTranslatef(bx, 0, 0);

glColor3ub(255, 0, 0);
glBegin(GL_POLYGON);
glVertex2d(410, 100);
glVertex2d(490, 100);
glVertex2d(485, 130);
glVertex2d(410, 130);
glEnd();

glColor3ub(255, 0, 0);
glBegin(GL_POLYGON);
glVertex2d(420, 130);
glVertex2d(475, 130);
glVertex2d(465, 160);
glVertex2d(430, 160);
glEnd();

// car window
glColor3ub(220, 220, 220);
glBegin(GL_POLYGON);
glVertex2d(425, 130);
glVertex2d(445, 130);
glVertex2d(445, 150);
glVertex2d(430, 150);
glEnd();

// car window
glColor3ub(220, 220, 220);
glBegin(GL_POLYGON);
glVertex2d(450, 130);
glVertex2d(470, 130);
```

```
    glVertex2d(465, 150);
    glVertex2d(450, 150);
    glEnd();

    // car wheel
    glColor3ub(0, 0, 0);
    circle(10, 14, 435, 100);
    circle(10, 14, 465, 100);

    glColor3ub(245, 245, 245);
    circle(6, 10, 435, 100);
    circle(6, 10, 465, 100);

    glPopMatrix();

    drawSecondCar(); // Draw the second car
    drawThirdCar();  // Draw the third car

    bx += .05;
    if (bx > 0)
        bx = -500;

    // Check for face-to-face with the second car
    if (bx + 490 > cx2 + 410 && bx + 410 < cx2 + 490 && secondCarLane == 0) {
        if (bx + 440 > cx2 + 410 && bx + 410 < cx2 + 490) {
            secondCarLane = 1; // Move to left lane
        }
    }

    glutPostRedisplay();

    glFlush();

    glutSwapBuffers();
}

void keyboardFunc(unsigned char key, int x, int y) {
    switch (key) {
        case 'a': // Move the main car to the left
            bx -= 5;
```

```
            break;
        case 'd': // Move the main car to the right
            bx += 5;
            break;
        case ' ': // Switch lanes for the second car
            secondCarLane = 1 - secondCarLane;
            break;
        default:
            break;
    }
}

void updateSecondCar(int value) {
    cx2 += dx2;
    if (cx2 < -100 || cx2 > 600)
        dx2 = -dx2;

    glutTimerFunc(10, updateSecondCar, 0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1000, 600);
    glutInitWindowPosition(300, 50);
    glutCreateWindow("A Moving Car Scenario");
    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboardFunc);
    glutTimerFunc(10, updateSecondCar, 0);
    glutMainLoop();
    return 0;
}
```